A Data Logger System Report in the Rocket Project

By: 4AEPD001 Khamphong Khongsomboon
Advisor: Assoc. Prof. Dr. Naohiko Shimizu

Department of Communications Engineering,
School of Information Technology and Electronics,
Tokai University.
2004-2005

Contents

A Data Logger System

1. Introduction

The Rocket Project is a project for development embedded systems, software, technical education and teaching material with record GPS data to find orbit trace of the rocket. Most of all the development embedded systems we do not have detail of the development and process therefore we can not get a chance for skill up from elementary to higher based on a real development and process. So this project is accommodating enforce the development experiment with multiple engineering and managers. We have to record the progress and then make a teaching material for development education and embedded systems.

2. Experiment requirement

In the experiment technical used record data when the rocket launch out to find the maximum height of the rocket so we need a recorder for record data from GPS with serial interface. The recorder we call Data Logger System or Data Logger Circuit. The experiment provided to put the circuit inside the rocket body before launch out. In this case the circuit must be fit size less than 30x30mm because diameter of the rocket is only 38mm, it should be light that mean the weight of the circuit must be less than 40g and also it must have waterproof and floating box for prevent the water flow inside and float when the rocket fall down to the sea.

The proposed of the circuit experiment:

- Used controller record orbit data from GPS to stoker data. The system must record GPS data 1 packet per 1second, in a GPS data packet has several data include position data, date/time data, GPS satellite information data and error index information data so we need to receive all of GPS data therefore transmit that data to PC for analysis and find the maximum height of the rocket fly out.
- Observation orbit of rocket from ground by 3 angles, the ground record can calculate the maximum height of the rocket to implement the quality software to the system.

3. The Data Logger System

In the data recorder or Data Logger System we need a good performance system for record data from GPS correctly and power supply can supply to the system as long as possible. The Data Logger System, main objective of this system used to be a recorder for record the rocket orbit and at the same time recorded the maximum height data from ground to the rocket. The Data Logger System consists of several sections as show in figure 1 show the Data Logger System diagram. It has several sections to combine a system such as: Power supply section, GPS section, Controller section, Store data section, Radio wave section and Transmission-Reception data to PC section.

3.1. Power supply section

According to GPS specification and period time for store data the GPS need 3.3V and 88mA when operation in continuous mode, therefore the power supply of system must be 3.3V and can supply to system more than 30minutes at the current between 150mA-200mA because the GPS sink current a lot, spend long time for catch the satellite and need a few

minute for preparation the rocket launch and also feed current to every section of the Data Logger System.

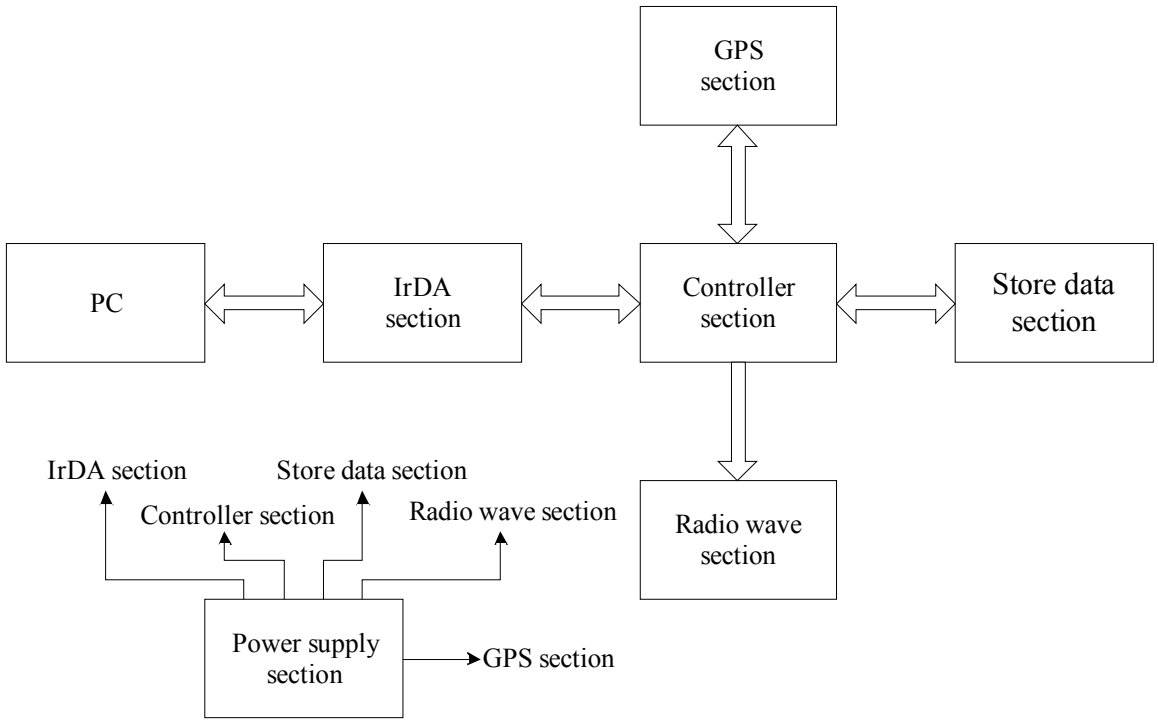


Figure1: Show the Data Logger System diagram

3.2. Controller section

The controller section is a microprocessor there is a main control of the system and also a center interface of a section to another section such as receive data from GPS section than transmit data into store data section, receive command from PC and than control the process as that command and etc. This microprocessor use programming for interface and control the system so it is the most importance of the system. In the development software, the programming function we must define every sequence process to enforce the system work and the system should have build-in test and hardware self check to confirm the system is ready start. If the test is fail it must be show alarm and if the test is passed the system go on to next process or wait for start system.

Consider the microprocessor specification will depend on the product in this project we select ATmega16L of ATmel cooperation product and it is 8-bit Microcontroller with 16K Bytes In-System Programmable Flash:

Features

• High-performance, Low-power AVR® 8-bit Microcontroller
• Advanced RISC Architecture
– 130 Powerful Instructions – Most Single-clock Cycle Execution
– 32 x 8 General Purpose Working Registers
– Fully Static Operation
– Up to 16 MIPS Throughput at 16 MHz
– On-chip 2-cycle Multiplier

• Nonvolatile Program and Data Memories
– 16K Bytes of In-System Self-Programmable Flash
Endurance: 1,000 Write/Erase Cycles
– Optional Boot Code Section with Independent Lock Bits
In-System Programming by On-chip Boot Program
True Read-While-Write Operation
– 512 Bytes EEPROM
Endurance: 100,000 Write/Erase Cycles
– 1K Byte Internal SRAM
– Programming Lock for Software Security
• JTAG (IEEE std. 1149.1 Compliant) Interface
– Boundary-scan Capabilities According to the JTAG Standard
– Extensive On-chip Debug Support
– Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
• Peripheral Features
– Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
– Real Time Counter with Separate Oscillator
– Four PWM Channels
– 8-channel, 10-bit ADC
8 Single-ended Channels
7 Differential Channels (TQFP Package Only)
2 Differential Channels with Programmable Gain (1x, 10x, 200x) (TQFP Package Only)
– Byte-oriented 2-wire Serial Interface
– Programmable Serial USART
– Master/Slave SPI Serial Interface
– Programmable Watchdog Timer with Separate On-chip Oscillator
– On-chip Analog Comparator
• Special Microcontroller Features
– Power-on Reset and Programmable Brown-out Detection
– Internal Calibrated RC Oscillator
– External and Internal Interrupt Sources
– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
• I/O and Packages
– 32 Programmable I/O Lines
– 40-pin PDIP and 44-lead TQFP
• Operating Voltages
– 2.7 - 5.5V (ATmega16L)
– 4.5 - 5.5V (ATmega16)
• Speed Grades
– 0 - 8 MHz (ATmega16L)
– 0 - 16 MHz (ATmega16)

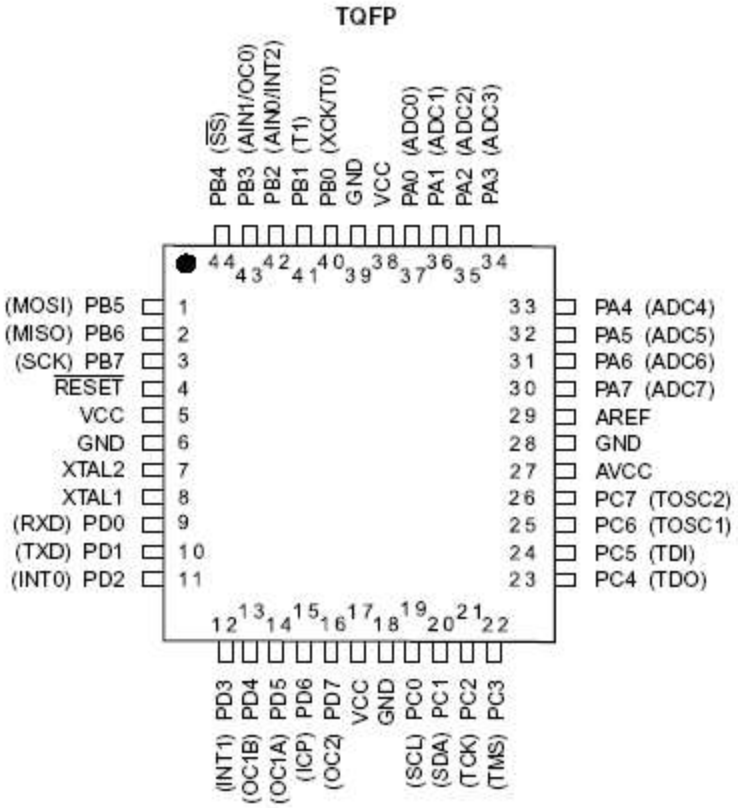


Figure 2: Pinouts ATmega16

3.2.1. I/O port

Make sure that you have enough I/O port for interface or access to the external section. Consider the initial setup and be long to schematic circuit connection.

3.2.2. Interface

Must be fast enough and specified as being able to operate at a certain maximum baud rate. The serial interface has several interface as SPI (Serial Peripheral Interface) interface, USART (Universal Asynchronous and Synchronous serial Receiver and Transmitter) interface and etc.

3.2.2.1. The SPI Interface

The SPI (Serial Peripheral Interface) interface that allows high-speed synchronous data transfer. Before interface the SPI must have initial setup.

- Initial SPI interface subroutine.

-Step 1: Setup MOSI (Master-Out Slave-In), SCK (Signal Clock) and CS (Chip Select) pin are output be long to ports name.

-Step 2: Set SPCR with 0101 0000 or 0x50

-Step 3: Return subroutine

Show some parameter in SPCR (SPI Control Register)

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 - SPIE: SPI Interrupt Enable
Set this bit is zero causes the SPI interrupt is not need to be executed.
• Bit 6 - SPE: SPI Enable
When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.
• Bit 5 - DORD: Data Order
When the DORD bit is written to one, the LSB of the data word is transmitted first. When the DORD bit is written to zero, the MSB of the data word is transmitted first.
• Bit 4 - MSTR: Master/Slave Select
This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If SS is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI master mode.
• Bit 3 - CPOL: Clock Polarity
Writing the CPOL is zero, SCK is low when idle.
• Bit 2 - CPHA: Clock Phase
The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK.
• Bits 1,0 - SPR1, SPR0: SPI Clock Rate Select 1 and 0
These two bits control the SCK rate of the device configured as a master. SPR1 and SPR0 have no effect on the slave. The relationship between SCK and the Oscillator Clock frequency fosc. When set SPR1 and SPR0 is zero, the SCK will be equal to fosc/4 at the same time SPI2X will be clear.

- The SPI transmission subroutine

When configured as a Master, the SPI interface has no automatic control of the SS line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the 8 bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI interrupt enable bit (SPIE) in the SPCR register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, SS line. The last incoming byte will be kept in the buffer register for later use.

The programming process of SPI transmission subroutine

-Step 1: Move data into SPDR
-Step 2: Polling until SPIF flag in SPSR register is set that is transmission data complete.
-Step 3: Return subroutine

Show some parameter in SPSR (SPI Status Register)

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 - SPIF: SPI Interrupt Flag

When a serial transfer is complete, the SPIF flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If SS is an input and is driven low when the SPI is in master mode, this will also set the SPIF flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI Data Register (SPDR).

- The SPI reception subroutine

This process used the same routine with SPI transmission process but when a serial transfer is complete, finally reading data in SPDR.

The programming process of SPI reception subroutine

-Step 1: Move a Dummy data into SPDR
-Step 2: Polling until SPIF flag in SPSR register is set
-Step 3: Read data in SPDR register that is reception data complete
-Step 4: Return subroutine

3.2.2.2. The USART interface

- Initial USART interface subroutine

-Step 1: Setup U2X flag in UCSRA is set for double speed
-Step 2: Set UBRR with 51 for set Baud rate at 9600bps
-Step 3: Set RXCIE, RXEN and TXEN flag in UCSRB register to one for enable interrupt receiver, enable receiver and transmitter.
-Step 4: Set UMSEL, UCSZ1 and UCSZ0 flag in UCSRC register to one for set asynchronous mode operation, format frame, 8bit data and 1stop bit.
-Step 5: Return subroutine

As bellow show initial setup the USART register parameter

- Setup Double Speed Operation (U2X)

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

The transfer rate can be doubled by setting the U2X bit in UCSRA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the receiver

will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the transmitter, there are no downsides.

• Bit 1 - U2X: Double the USART transmission Speed:

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation. Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- Setup Baud Rate

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

As the specification of GPS used 9600bps serial interface and the equations in table1 we will have the USART Baud Rate Register (UBRR)

$$UBRR = \frac{f_{osc}}{8BAUD} - 1$$

$$UBRR = \frac{4MHz}{8x9600} - 1$$

$$UBRR = 51$$

Where, the system clock frequency is 4MHz

Table 1: Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

– **Enable receiver, transmitter and interrupt receiver**

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 - RXCIE: RX Complete Interrupt Enable
Writing the RXC flag to one for enables Receive Complete interrupt. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the global interrupt flag in SREG is written to one and the RXC bit in UCSRA is set.
• Bit 6 - TXCIE: TX Complete Interrupt Enable
Do not need TX interrupt enable so writing this bit to zero
• Bit 5 - UDRIE: USART Data Register Empty Interrupt Enable
Do not need UDRIE interrupt enable so writing this bit to zero
• Bit 4 - RXEN: Receiver Enable
Writing this bit to one enables the USART receiver. The receiver will override normal port operation for the RxD pin when enabled. Disabling the receiver will flush the receive buffer invalidating the FE, DOR and PE flags.
• Bit 3 - TXEN: Transmitter Enable
Writing this bit to one enables the USART transmitter. The transmitter will override normal port operation for the TxD pin when enabled. The disabling of the transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit shift register and transmit buffer register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.
• Bit 2 - UCSZ2: Character Size
Writing UCSZ2 bits is zero for sets the number of data 8 bits (character size) in a frame the receiver and transmitter use.
• Bit 1 - RXB8: Receive Data Bit 8
When receive serial frames with 8 data bits must be set this bit is zero
• Bit 0 - TXB8: Transmit Data Bit 8
When transmit serial frames with 8 data bits must be set this bit is zero

- Setup frame format data

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

• Bit 7 - URSEL: Register Select
This bit selects between accessing the UCSRC or the UBRRH register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.
• Bit 6 - UMSEL: USART Mode Select
This bit selects between asynchronous and synchronous mode of operation. Therefore set this bit is zero for asynchronous mode.

• Bit 5:4 - UPM1:0: Parity Mode

Writing this bit is zero for non parity bit within data frame.

• Bit 3 - USBS: Stop Bit Select

This bit selects the number of stop bits to be inserted by the transmitter. Set this bit is zero for 1 stop bit.

• Bit 2:1 - UCSZ1:0: Character Size

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (character size) in a frame the receiver and transmitter use. Set these bit are 1 that mean set 8 bit data in a frame.

• Bit 0 - UCPOL: Clock Polarity

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

- The USART transmission subroutine

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The processor can load the transmit buffer by writing to the UDR (USART I/O Data Register) I/O location. The buffered data in the transmit buffer will be moved to the shift register when the shift register is ready to send a new frame. The shift register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the shift register is loaded with new data, it will transfer one complete frame at the rate given by the baud register, U2X bit or by XCK depending on mode of operation. The USART transmit function based on polling of the *Data Register Empty* (UDRE) flag. When using frames with less than eight bits, the most significant bits written to the UDR are ignored.

The programming process of a data transmission

-Step 1: Polling until UDRE flag in UCSRA is set

-Step 2: Move data in UDR, there is finished transmission data 1 byte.

-Step 3: Return subroutine

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

• Bit 5 - UDRE: USART Data Register Empty

The UDRE flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE flag can generate a Data Register Empty interrupt (see description of the UDRIE bit). UDRE is set after a reset to indicate that the transmitter is ready.

- The USART Reception subroutine

The receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the receive shift register until the first stop bit of a frame is received. A second stop bit will be ignored by the receiver. When the first stop bit is received, i.e., a complete serial frame is present in the receive shift register, the contents of the shift register will be moved into the receive buffer. The

receive buffer can then be read by reading the UDR I/O location. The USART receive function based on polling of the Receive Complete (RXC) flag. Then move data from the UDR into SRAM which prepare for the next process.

The programming process of a reception data

-Step 1: Polling until RXC flag in UCSRA is set that mean receive data complete. In this point it might receive data every time when RXC flag is set because we set RXCIE flag in the UCSRB register of USART reception subroutine for interrupt enable if receive data complete.

-Step 2: Move data from UDR to SRAM.

-Step 3: Check SRAM address equal to limited address (from 0x0060 to 0x0158). If it is equal the address return to start address at 0x0060 again if it is not equal continuous store data until to 0x0158.

-Step 4: Return subroutine

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

• Bit 7 - RXC: USART Receive Complete

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

3.2.3. Memory

The memory is a key part of embedded system design you must select an unnecessarily solution but do need to ensure that you have enough memory. The memory has two part: RAM is the sum of all internal buffer at 0x0000 to 0x005F, FIFOs (first in, first out) for store data has address at 0x0060 to 0x0158 and stacks pointer has address at 0x045F, and ROM is the sum of the program code and any ROM-based tables.

3.2.4. Interrupt

The interrupt are used to notify the processor of special events such as a timer that timed out or a piece of hardware that needs attention. Counting the events that need interrupts is straightforward, but be sure to take into account internal interrupt sources as well. An interrupt can notify the processor when has new data, when a timer rolls over or when almost any asynchronous event happens.

In this case we used interrupt vector when an interrupt is acknowledge. The interrupt vector tell the processor where to go to service the interrupt. Timer/Counter is an interrupt vector to execute when the time rolls over or compare is equal.

The Timer/Counter2 is a general purpose, single channel, 8bit Timer/Counter module. We used the feature is CTC clear timer on compare match (Auto reload) and also compare match interrupt source (OCF2).

- Initial setup Timer/Counter2 subroutine

-Step 1: Set WGM2, CS21 and CS20 flag in TCCR2 register for CTC mode and fclk/32

-Step 2: Set OCF2 flag in TIFR register for output compare mode

-Step 3: Set OCIE2 flag in TIMSK register for enable compare match interrupt

-Step 4: Move OCR2 with 142 for set interrupt frequency

-Step 5: Return subroutine

3.3. GPS section

GPS has several operation modes and has a lot of options for setup before transmission-reception data there is belong to system requirement and user need. The GPS has standard specifications as bellow:

- Module Power Supply Voltage: 3.3V
- The maximum current draw in continuous operating: 88mA
- System: Full-duplex asynchronous
- Speed: 9600bps (Transfer data by serial communication)
- Start bit: 1bit
- Data length: 8bit
- Stop bit: 1bit
- Parity bit: None

In the operation mode the Rocket Project provide to use Full-Time Mode for transmit data 124byte per second, this data is called a packet of GPS data or Periodical Data Output, there are includable:

- Position data output (23-byte fixed length) include: Latitude, Longitude and Altitude.
- Date/Time data output (10-byte fixed length)
- GPS Satellite Information Output (76-byte fixed length)
- Error Index Information Output (15-byte fixed length)

The commands code for setup Full-Time Mode operation is

- Header 0xC0
- Setup command 0xA3
- Output data items 0x8F
- Checksum 0xEC
- Terminator 0xCA

3.4. Store data section

This section use for store data from GPS and all of the data must be still stay either has supply or not. In this case EEPROM is suitable for store exist data until rewrite new data, EEPROM store data by 1byte into 1 address. Reference to the GPS transmission data before write into EEPROM 124byte per second and as a requirement need to have 2byte header synchronize pattern and 2byte for terminator, therefore the data packet or 1 frame is 128byte will store into EEPROM in a second. The store data process, first store START command 1 frame packet than store data packet until has interrupt stop command to stop system and then store STOP command 1 frame packet the system stop record data, as show in figure2 show the store packet of command and data into EEPROM.

If the rocket launch about 1minute the data must be here:

1 sec = 124byte this is GPS packet

1 sec = 128byte this is EEPROM packet

1minute = 7.5kbyte

So the system needed the EEPROM able to store data more than 7.5kbyte and still have data when has or doesn't have power supply. The store process before record data into EEPROM must wait the system found 3D position-fixing and occurred more than 5 satellites or equal then start record data. If the EEPROM is full the system must stop record data automatic by itself. And another once importance when put the circuit into waterproof box it must spend a few minute so if the store data section is small, it will be full before the rocket launch out.

Format 1 frame (128byte) of data packet from GPS write into EEPROM

Synchronous Pattern of correct data 2 byte	Data packet from GPS 124 byte	Cyclic Redundancy Check (CRC) 2 byte

Syn. Pat. 0xE0	Syn. Pat. 0xE0	Data packet from GPS 124 byte	Checksum 1byte	Invert-Checksum 1 byte

Format 1 frame (128byte) of error data packet from GPS write into EEPROM

Synchronous Pattern of error data 2 byte	Data packet from GPS 124 byte	Cyclic Redundancy Check (CRC) 2 byte

Syn. Pat. 0xE8	Syn. Pat. 0xE8	Error Data packet from GPS 124 byte	Checksum 1byte	Invert-Checksum 1 byte

Format 1 frame (128byte) of command write into EEPROM

Synchronous Pattern of command 2 byte	Commands + Empty 124 byte	Cyclic Redundancy Check (CRC) 2 byte

Syn. Pat. 0xE4	Syn. Pat. 0xE4	Command 1byte	Empty or Zero 123 byte	Checksum 1byte	Invert-Checksum 1 byte

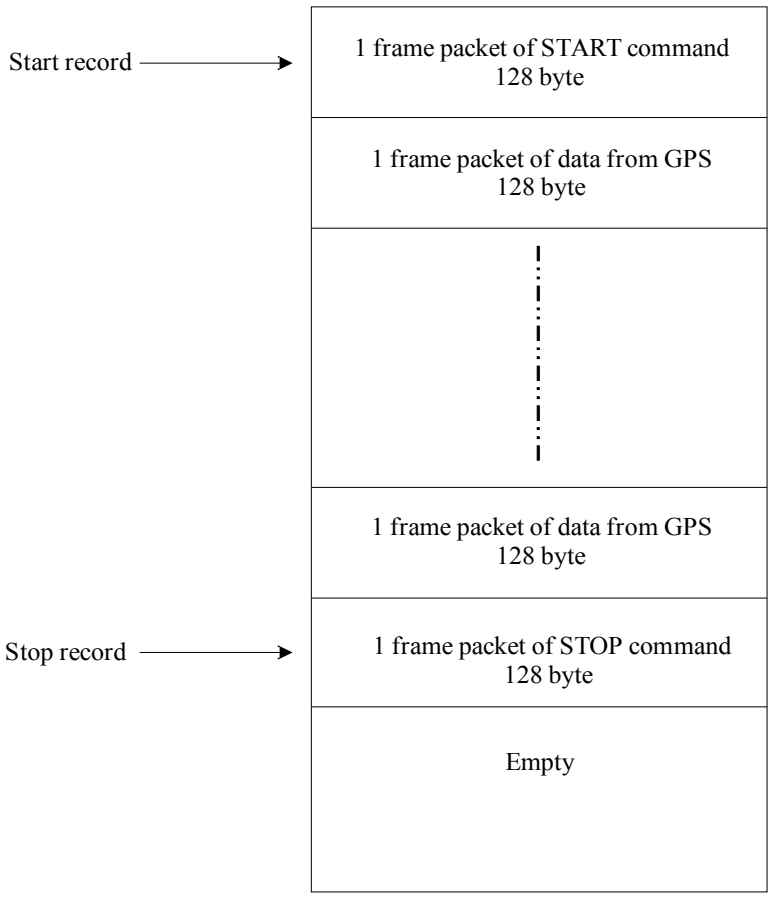


Figure3 show the store packet of command and data into EEPROM.

Timing Diagrams (for SPI Mode 0 (0,0))

Synchronous Data Timing

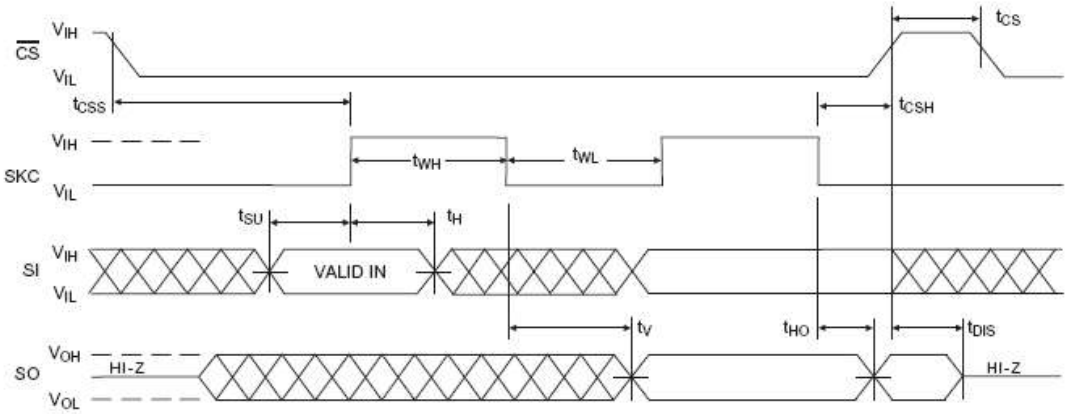


Figure4: Show the synchronous data timing of EEPROM

3.4.1. The EEPROM specification

We used The AT25HP256/512 provides 262,144/524,288 bits of serial electrically erasable programmable read only memory (EEPROM) organized as 32,768/65,536 words of 8-bits each. The device is optimized for use in many industrial and commercial applications where high-speed, low-power, and low-voltage operation are essential. The AT25HP256/512 is designed to interface directly with the synchronous serial peripheral interface (SPI) of the 6800 type series of microcontrollers as in Figure3: Show the synchronous serial peripheral interface timing of EEPROM. The AT25HP256/512 utilizes an 8-bit instruction register. The list of instructions and their operation codes are contained in Table 2. All instructions, addresses, and data are transferred with the MSB first and start with a high-to-low CS transition.

Table2: Instruction Set for the AT25HP256/512

Instruction Name	Instruction Format	Operation
WREN	0000 X110	Set Write Enable Latch
WRDI	0000 X100	Reset Write Enable Latch
RDSR	0000 X101	Read Status Register
WRSR	0000 X001	Write Status Register
READ	0000 X011	Read Data from Memory Array
WRITE	0000 X010	Write Data to Memory Array

The Table 3 show the Read Status Register instruction provides access to the status register. The READY/BUSY and Write Enable status of the device can be determined by the RDSR instruction. Similarly, the Block Write Protection bits indicate the extent of protection employed. These bits are set by using the WRSR instruction. The three bits, BP0, BP1, and WPEN are nonvolatile cells that have the same properties and functions as the regular memory cells (e.g. WREN, tWC, RDSR).

Table3: Status Register Format

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WPEN	X	X	X	BP1	BP0	WEN	$\overline{RDY}$

Table4: Read Status Register Bit Definition

Bit	Definition
Bit 0 ($\overline{RDY}$)	Bit 0 = 0 ($\overline{RDY}$) indicates the device is READY. Bit 0 = 1 indicates the write cycle is in progress.
Bit 1 (WEN)	Bit 1= 0 indicates the device *is not* WRITE ENABLED. Bit 1 = 1 indicates the device is WRITE ENABLED.
Bit 2 (BP0)	See Table 4.
Bit 3 (BP1)	See Table 4.
Bits 4-6 are 0s when device is not in an internal write cycle.	
Bit 7 (WPEN)	See Table 5.
Bits 0-7 are 1s during an internal write cycle.	

3.4.2. The EEPROM interface

The interface between microprocessor to EEPROM and depend on EEPROM used serial interface specification therefore we used SPI serial interface for transfer data to EEPROM. The transfer data process has write data into EEPROM process and read data from EEPROM process. So we can separate on the programming process.

First, set write enable to EEPROM by:

- Set CS high-to-low
- Transmit 0000 X110 or 0x06 to EEPROM by SPI transmission subroutine
- Set CS low-to-high

Next, read the status register

- Set CS high-to-low
- Transmit 0000 X101 or 0x05 to EEPROM by SPI transmission subroutine
- Transmit 0000 0000 to EEPROM by SPI transmission subroutine for receive status register
- Compare the status register equal to 0000 0010 or 0x02. If equal that mean the Device is WRITE ENABLE
- Set CS low-to-high

Then, write data into EEPROM with 128byte or page write

- Set CS high-to-low
- Transmit write instruction 0000 0010 or 0x02 by SPI transmission subroutine
- Transmit higher address of EEPROM by SPI transmission subroutine
- Transmit lower address of EEPROM by SPI transmission subroutine
- Loop 2 times for transmit 2 byte data code synchronous pattern
- Read GPS data in SRAM
- Transmit data from SRAM to EEPROM 124byte by SPI transmission subroutine
- Transmit checksum
- Transmit invert checksum
- Set CS low-to-high

3.5. Radio wave section

This is the FM radio wave propagation in the air space, it has range between 76 to 90 MHz (used in Japan) if we provide the FM radio frequency in this range it is very easy for receive by radio receiver. The proposed of the FM radio wave propagation to inform the base station when the system found 3D position-fixing and occurred more than 5 satellite or equal by change the frequency. In addition used the FM radio wave propagation to find position of the rocket when the rocket down to the sea.

The FM radio wave specification, used the sound frequency 440Hz modulate in FM radio wave. When the system start we set the sound frequency 2second sound-on and 2second sound-off after occurred the 3D position-fixing and more than 5 satellite the oscillator frequency will change 0.5second sound-on and 0.5second sound-off therefore the system start record data.

When set an interrupt is 2.27ms = 440Hz. So the hexadecimal code of sound frequency is:

- 0.5sec.-on and 0.5sec.-off = 1sec. the interrupt will count 440 times equal to 0x01B8
- 2sec.-on and 2sec.-off = 4 sec. the interrupt will count 1760 times equal to 0x06E0

In the programming process we used Timer/Counter2 in CTC clear timer on compare match (Auto reload) mode operation for make oscillator:

- Wait for interrupt
- Count interrupted 4seconds.
- Set port for sound on and off, there is change a time
- Loop this value until the system occurred 3D position-fixing and 5 satellite will change oscillator frequency to 1sec.
- Count interrupted 1second.
- Set port for sound on and off, there is change a time
- Loop here forever.

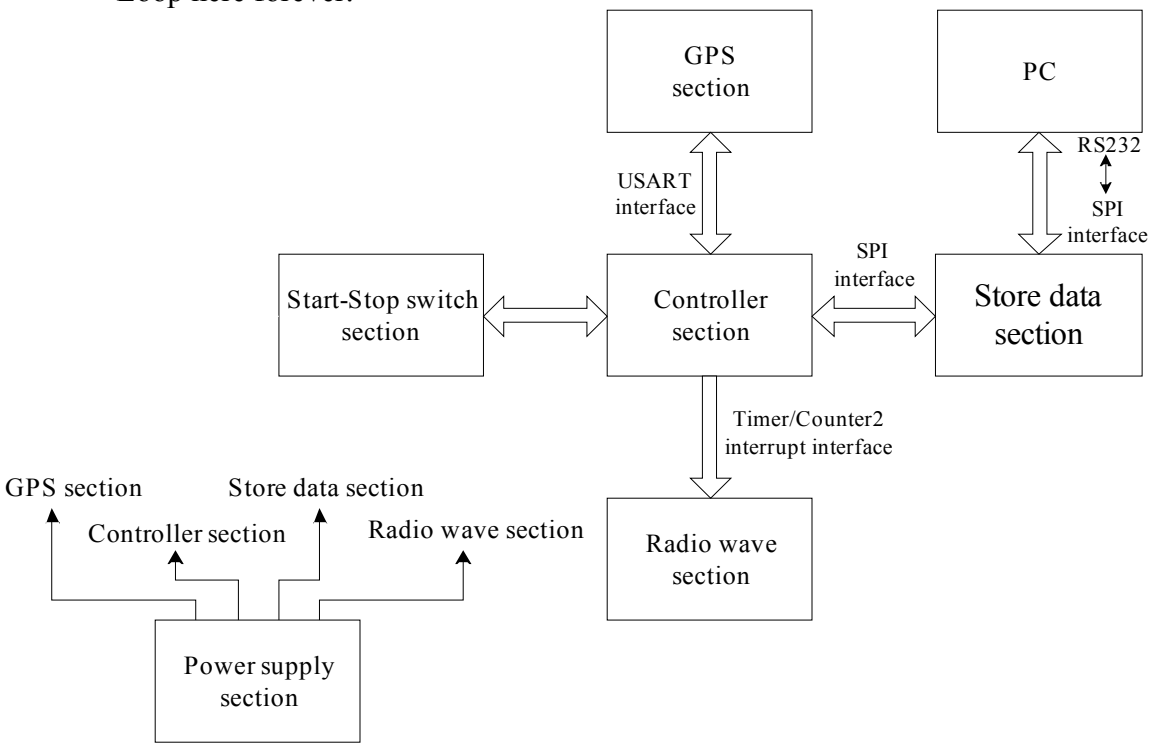


Figure5: Show the Data Logger System diagram spare part

3.6. Transmission-Reception data to PC section

It is interfacial between the Data Logger Circuit to PC by using IrDA serial communication, where this interface has two parts: PC transmission commands code and reception data from the Data Logger Circuit. The commands code include start command, stop command, request data commands, acknowledge command and no acknowledge command, all of these commands are only 1byte hexadecimal number. For the reception data, the circuit transmit data to PC the format frame of data should has synchronous pattern and data, it is quite comfortable for separate frame of data.

- Start command used for start the system
- Stop command used for stop the system
- Request command used for request the controller section to transmit data to PC
- Acknowledge command used for response the system ready
- No acknowledge command used for response the system not ready or error

Format transmission data packet from EEPROM to PC

<SOH>	<BlockNO>	<~BlockNO>	<EEPROM Data 1 Frame>	<Check Sum>
1 byte	1 byte	1 byte	Data 128 byte	1 byte

In fact, the IrDA interface is very difficult to interface so we change to used a spare method such as use push-switch for start-stop system and read data directly from EEPROM by PC as show in figure 3 show the Data Logger System diagram spare part.

4. Program structure and subroutine

4.1. The Main program

In a program should has the main program for descript the proposed and process of the system. As the flowchart 1 is show the main program flow when power-on for start system.

4.1.1. Initial setup process

-Step 1: Setup initial I/O port and be long to circuit diagram, set stack pointer address at 0x045F, set initial SPI interface subroutine and set initial USART interface subroutine.

-Step 2: Set initial START and STOP switch are high status (high value=supply voltage)

-Step 3: Set Initial oscillator frequency by set the first count value for Timer/Couonter2 interrupt with 2second sound-on and 2second sound-off.

-Step 4: Set GPG power-on by set a port of microprocessor

-Step 5: RCALL Delay time 1sec. subroutine 5 times (5 seconds)

-Step 6: Setup GPS output by RCALL **Periodical data output subroutine**

4.1.2. Main routine process

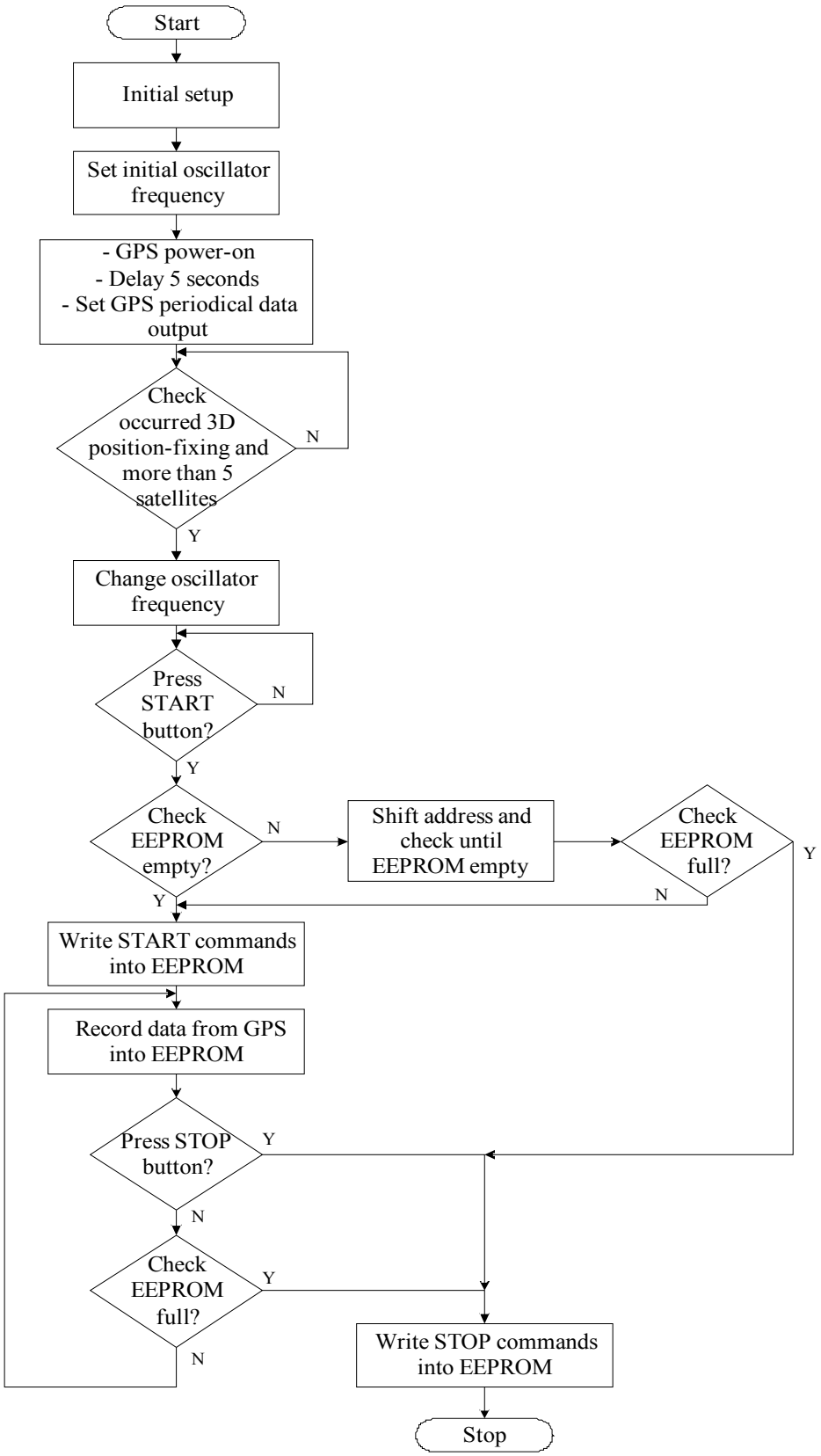
-Step 7: RCALL **Check 3D position-fixing and more than 5 satellite subroutine** for waited data and check position data packet until GPS data occurred 3D position-fixing and more than 5 satellite which it might change FM oscillator frequency to 0.5second sound-on and 0.5second sound-off by **Timer/Counter2 interrupt subroutine**.

-Step 8: Check START switch it must be still waited here until START switch press then jump to **check EEPROM emptied subroutine**. If the EEPROM is empty there is continuous write START command in to EEPROM by **write START command into EEPROM subroutine** and if the EEPROM is not empty the check empty routine will shift to next address until found empty address or if EEPROM is full the check empty routine must stop system.

-Step 9: RCALL **Write data into EEPROM subroutine** after pressed START button the system will start record data into EEPROM. Here it has to **check GPS data packet subroutine** there are correct or incorrect data and separate data format therefore write into EEPROM.

-Step 10: The system might automatic stop record data when STOP switch was pressed or EEPROM is full and before the system stop must used RCALL **write STOP command into EEPROM subroutine**.

-Step 11: Stop the system.

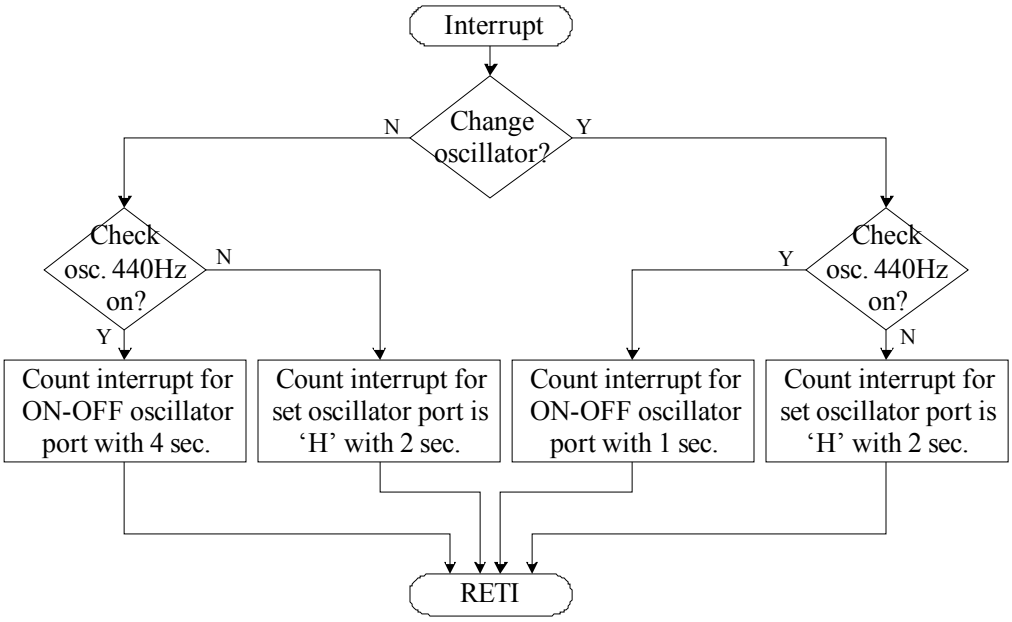


Flowchart 1 is show the main program flow

4.2. The subroutine program

4.2.1. Timer/Counter2 interrupt subroutine

This subroutine is classified into **Set initial oscillator frequency** and **Change oscillator frequency** see in the flowchart 1. Both of these boxes include in the **Timer/Counter2 interrupt subroutine** as show in the flowchart 2 show the way of **Timer/Counter2 interrupt subroutine** how to the oscillator change with the frequency 440Hz.



Flowchart 2: Show the condition of the oscillator change by Timer/Counter2 interrupt routine

The programming process

-Step1: Check if the oscillator is not changeless goes to next step and if it is changeless go to step 5
-Step 2: Set the oscillator port ON-OFF with frequency 440Hz at 4 seconds
-Step 3: Set the oscillator port 'H' or ON with 2 seconds
-Step 4: Return step 1
-Step 5: Set the oscillator port ON-OFF with frequency 440Hz at 1 seconds
-Step 6: Set the oscillator port 'H' or ON with 2 seconds
-Step 7: Return step 5

4.2.2. Check 3D position-fixing and more than 5 satellite subroutine

There is checked 3D position-fixing with check bit 2 in byte 2nd of position data packet if this bit is set that mean 3D position-fixing are occurred and if this bit is clear, there is not occurred. For check more than 5 satellites is checked byte 21st must be equal or more than 0x05 in the position data packet.

The programming process:

-Step 1: Read data in SRAM and check header of position data packet is equal to 0xE0 or not. If not return to step1 again and if correct goes to next step

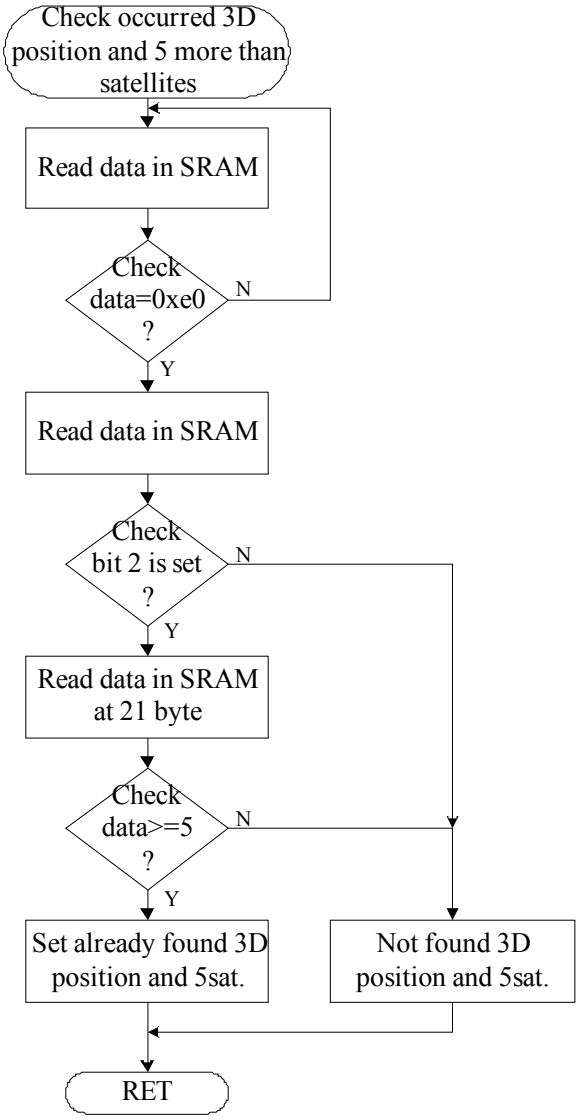
-Step 2: Check the next data which bit 2nd is set or not. If set goes to next step and if clear return the routine.

-Step 3: Check the data at 21st byte of the position data packet if this byte is equal to or more than 0x05 goes to next step and if it is not goes to step5

-Step 4: Set already found 3D position-fixing and more than 5 satellites then go to step6

-Step 5: Not find 3D position-fixing and more than 5 satellites

-Step 6: Return subroutine



Flowchart 3: Check the occurred 3D position-fixing and more than 5 satellites routine

4.2.3. Check EEPROM emptied subroutine

-Step 1: Clear CS

-Step 2: Send read status register 0x05 to EEPROM by SPI transmission subroutine

-Step 3: Send DUMMY data 0x00 to EEPROM by SPI transmission subroutine for receive EEPROM status

-Step 4: If EEPROM status equal to 0x00 goes to next step and if it is not, return to step1

-Step 5: Send read instruction 0x03 by SPI transmission subroutine

-Step 6: Send index EEPROM high address by SPI transmission subroutine

-Step 7: Send index EEPROM low address by SPI transmission subroutine

-Step 8: Send DUMMY data 0x00 to EEPROM by SPI transmission subroutine for receive a data

-Step 9: Set CS

-Step 10: If that data is equal to 0x00 goes to step12 and if it is not, goes to next step

-Step 11: Increase EEPROM address with 0x80 and goes to step1

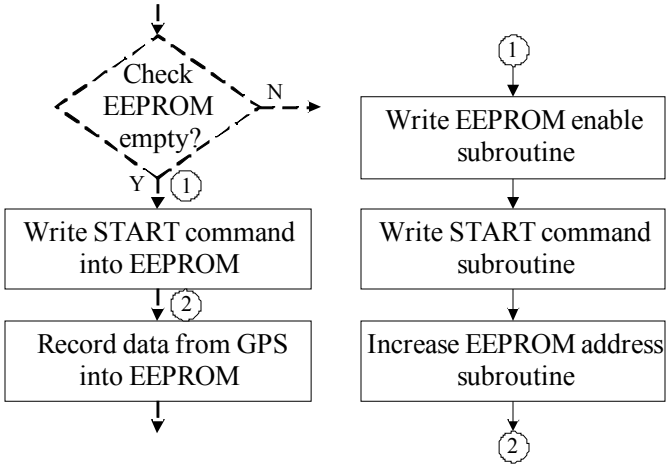
-Step 12: Return subroutine.

4.2.4. Write START command into EEPROM subroutine

-Step 1: RCALL write EEPROM enable subroutine

-Step 2: RCALL write START command subroutine

-Step 3: RCALL increase EEPROM address subroutine



Flowchart 4: Show the program flow in Write START command into EEPROM subroutine

4.2.5. Write STOP command into EEPROM subroutine

The step program flow is the same with **Write START command into EEPROM subroutine** only change START command code (0xF8) to STOP command code (0xF4)

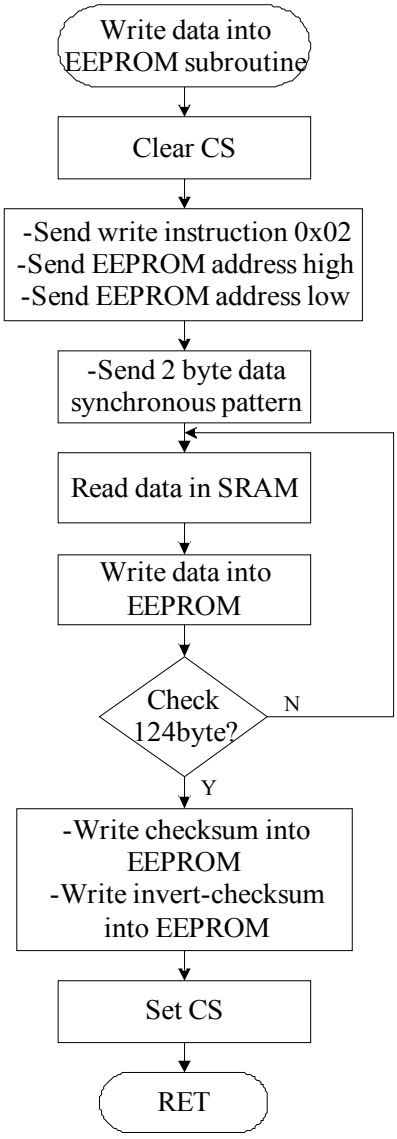
4.2.6. Write data into EEPROM subroutine

As in flowchart 5 show the flow of write data into EEPROM subroutine

-Step 1: Clear CS

-Step 2: Send write instruction 0x02 by SPI transmission subroutine

-Step 3: Send index EEPROM high address by SPI transmission subroutine
-Step 4: Send index EEPROM low address by SPI transmission subroutine
-Step 5: Send data synchronize pattern (0xE0 if the data is corrected and 0xE8 if the data is not corrected) 2byte into EEPROM by SPI transmission subroutine
-Step 6: Send data into EEPROM 124byte (page writing) by SPI transmission subroutine
-Step 7: Send checksum 1byte by SPI transmission subroutine
-Step 8: Send invert-checksum 1byte by SPI transmission subroutine
-Step 9: Set CS
-Step 10: Return subroutine

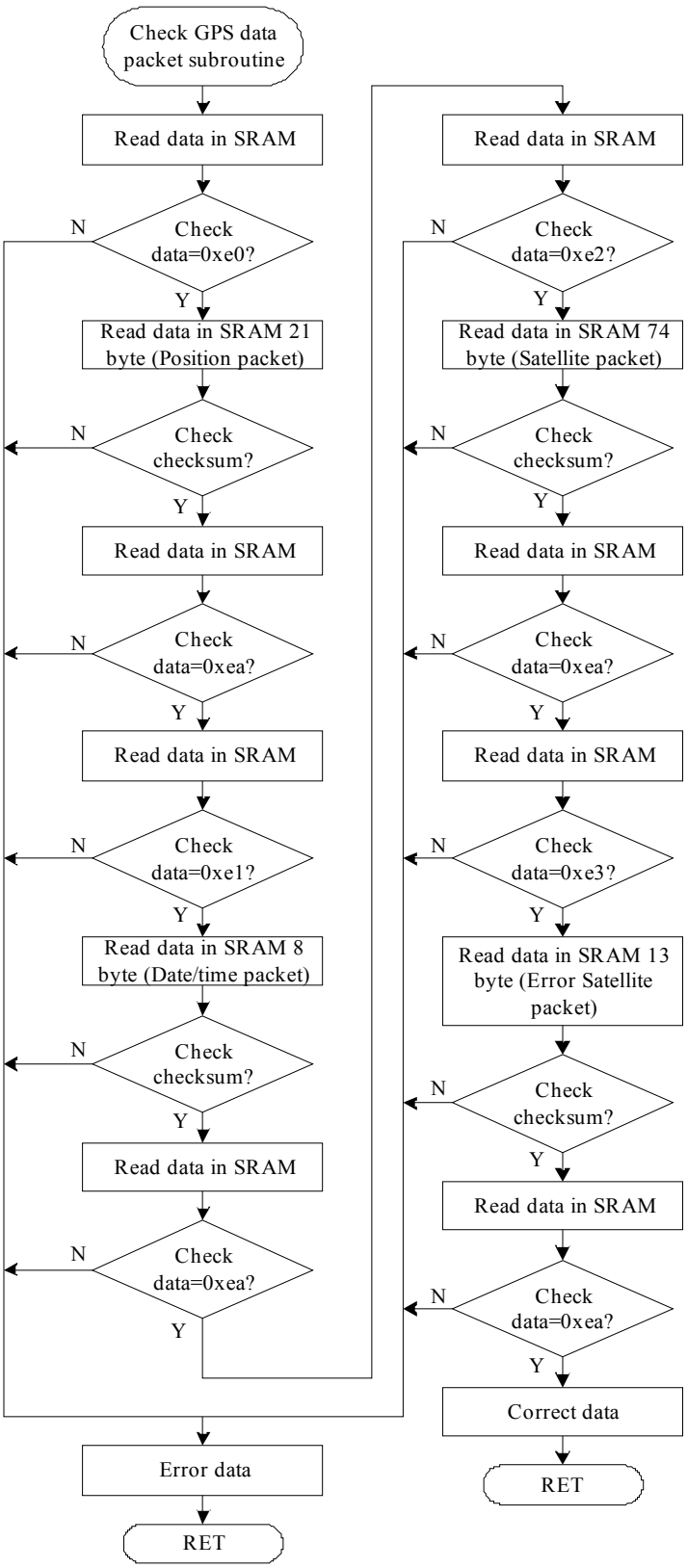


Flowchart 5: The write data into EEPROM subroutine process flow

4.2.7. Check GPS data packet subroutine

The check GPS data packet subroutine has programming process flow as in flowchart 6.

-Step 1: Read data (Header code of position data packet) from SRAM
-Step 2: Check header code packet 0xE0. If equal goes to next step and if not return to step1
-Step 3: Increase SRAM address 21 times and then read data in SRAM
-Step 4: Compare the checksum if equal goes to next step and if not goes to step26
-Step 5: Read data from SRAM
-Step 6: Check terminator code packet 0xEA. If equal goes to next step and if not return to step26
-Step 7: Read data (Header code of Data/Time data packet) from SRAM
-Step 8: Check header code packet 0xE1. If equal goes to next step and if not return to step26
-Step 9: Increase SRAM address 8 times and then read data in SRAM
-Step 10: Compare the checksum if equal goes to next step and if not goes to step26
-Step 11: Read data from SRAM
-Step 12: Check terminator code packet 0xEA. If equal goes to next step and if not return to step26
-Step 13: Read data (Header code of GPS satellite information data packet) from SRAM
-Step 14: Check header code packet 0xE2. If equal goes to next step and if not return to step26
-Step 15: Increase SRAM address 74 times and then read data in SRAM
-Step 16: Compare the checksum if equal goes to next step and if not goes to step26
-Step 17: Read data from SRAM
-Step 18: Check terminator code packet 0xEA. If equal goes to next step and if not return to step26
-Step 19: Read data (Header code of error index information data packet) from SRAM
-Step 20: Check header code packet 0xE3. If equal goes to next step and if not return to step26
-Step 21: Increase SRAM address 13 times and then read data in SRAM
-Step 22: Compare the checksum if equal goes to next step and if not goes to step26
-Step 23: Read data from SRAM
-Step 24: Check terminator code packet 0xEA. If equal goes to next step and if not return to step26
-Step 25: Set the GPS data packet is correct and goes to step 27
-Step 26: Set the GPS data packet is not correct
-Step 27: Return subroutine



Flowchart 6: Show the check GPS data packet subroutine process flow

4.2.8. Write EEPROM enable subroutine

-Step 1: Clear CS
-Step 2: Send write enable latch instruction 0x06 by SPI transmission subroutine
-Step 3 Set CS
-Step 4: Clear CS
-Step 5: Send read status register 0x05 to EEPROM by SPI transmission subroutine
-Step 5: Send DUMMY data 0x00 to EEPROM by SPI transmission subroutine for receive EEPROM status
-Step 6: If this status is equal to 0x02 goes to next step and if it is not, goes to step4
-Step 7: Return subroutine

4.2.9. Write START command subroutine

-Step 1: Clear CS
-Step 2: Send write instruction 0x02 by SPI transmission subroutine
-Step 3: Send index EEPROM high address by SPI transmission subroutine
-Step 4: Send index EEPROM low address by SPI transmission subroutine
-Step 5: Send synchronize pattern (0xE4) 2byte into EEPROM by SPI transmission subroutine
-Step 6: Send START command (0xF8) 1byte by SPI transmission subroutine
-Step 7: Send zero (0x00) 123byte by SPI transmission subroutine
-Step 8: Send checksum 1byte by SPI transmission subroutine
-Step 9: Send invert-checksum 1byte by SPI transmission subroutine
-Step 10: Set CS
-Step 11: Return subroutine

4.2.10. Increment EEPROM address subroutine

-Step 1: Add the EEPROM low address with 0x80
-Step 2: Compare low address equal to 0x00. If equal goes to step5 and if not goes to next step
-Step 3: Compare high address equal to 0xFF. If equal goes to next step and if it is not, goes to step 6
-Step 4: Set EEPROM status is full and goes to step6
-Step 5: Increase high address
-Step 6: Return subroutine

4.2.11. Setup GPS (Periodical data output) subroutine

-Step 1: Load header value 0xC0 into UDR register
-Step 2: CALL USART transmission routine
-Step 3: Load setup command 0xA3 into UDR register
-Step 4: CALL USART transmission routine
-Step 5: Load output data items 0x8F into UDR register
-Step 6: CALL USART transmission routine
-Step 7: Load checksum 0xEC into UDR register
-Step 8: CALL USART transmission routine
-Step 9: Load terminator 0xCA into UDR register
-Step 10: CALL USART transmission routine

4.2.12. Limited SRAM address subroutine

-Step 1: Compare SRAM high address, if it is less than 0x01, goes to step4 and if not goes next step

-Step 2: Compare SRAM low address, if it is equal to 0x58, goes to next step and if not goes step4

-Step 3: Clear SRAM address and return value at 0x0060

-Step 4: Return subroutine

4.2.13. Delay time 1sec.

-Step 1: RCALL Delay time subroutine 75msec..

-Step 2: Count 14 times if it is not equal goes to step1 and if equal goes to next step

-Step 3: Return subroutine.

Delay time 75msec subroutine.

-Step 1: RCALL Delay time 300usec subroutine.

-Step 2: Count 250 times if it is not equal goes to step1 and if equal goes to next step

-Step 3: Return subroutine.

Delay time 300usec subroutine.

-Step 1: Used no operation (nop) 1.5usec.

-Step 2: Count 240 times if it is not equal goes to step1 and if equal goes to next step

-Step 3: Return subroutine.

5. Conclusion

The Rocket Project is proposed of development the embedded systems, software engineering, technical education documents and teaching materials. So the research requirement we used the requirement and the specification of the system:

- Hardware requirement: small size (less than 30x30mm), light (weight less than 40g), waterproof and floating, power supply can use more than 30minutes at minimum current 150mA, set GPS in Full time mode operation and EEPROM must be suitable in period of store data time.
- Software requirement: definition of every function, self-test system if the system is fail should have alarm and if the system is passed go on to the next process.

In this document also descript the system specification, devices interface and programming main routine and subroutine of the Data Logger System which how can the system operate and record data process. There is very importance for used these document to improve and develop the embedded systems and software go on to high level in the near future. For the data in EEPROM we will upload and analyze in PC to find the maximum height of rocket fly out and than comparison with ground record it can calculate with 3 angles record.