Design and Implementation of Data Logger Using Lossless Data Compression Method for Internet of Things

Febrian Hadiatna*[1], Hilwadi Hindersah*[2], Desta Yolanda*[3], Muhammad Agus Triawan*[4]

*School of Electrical Engineering and Informatics, Bandung Institute of Technology
Ganesha Street 10, Bandung 40132, Indonesia

[1]febrianhadiatna@students.itb.ac.id
[2]hilwadi@lskk.ee.itb.ac.id
[3]destayolanda@students.itb.ac.id
[4]magustriawan@students.itb.ac.id

Abstract— **Internet of Things (IoT) has the ability to monitor and control various electronic devices. This technology need M2M device that serves as the middleware. The data logger can be developed into M2M devices on the IoT. These devices have limited external data storage memory, which can be useful as a data storage when a transmission failure due to limited network. In the monitoring process with a lot of sensors, data logger capable of transmitting and storing data in memory with limited data. The memory capacity of the data logger limited size can increase the size by using the method of data compression. Compression method used in this device is lossless. Lossless compression method used for data compression in the data logger. This method can restore the size of data that has been compressed to the size of the original. In this study, we obtained the compression ratio (CR) is 50% for the compression method used in the data logger. Time sampling that can be set in data logger is more than ± 1518ms**

Keywords— **IoT, Data Logger, Compression, Lossless, M2M**

I. INTRODUCTION

Internet of Things (IoT) has ability to monitor and control various electronic devices via online. When the infrastructure was poor, for example unstable internet networks, can cause problems in process of monitoring and control. On these problems required a system that can handle these conditions.

Data logger is designed to record data for a period of time[1]. Data logger has been used widely not only in electronic worlds but in all systems which relates to technology [2]. It is effectively used in chemical, biomedical, mechanical, electrical and industrial fields [1]. Real world data logger applications are typically not only recording the data, but also some combinations of online analysis, offline analysis, display, report generator, and data sharing [3].

Data logger implanted with IoT technology will be the solution to these problems. The data logger will serve as machine to machine (M2M) IoT. The results of the data acquisition process on each of the sensors used will be stored on external data storage memory and cloud. When bad network, data can not be transmitted to the cloud, but remain saved on the external data storage memory. Data logger will transmit all of data that has not been sent after a good network.

External data storage memory is an important part of the data logger. On the external data storage with capacity is limited, a method of data compression can help in increasing the amount of data.Data compression is a method of reducing the amount of memory required to store data by encoding it and minimizing redundancy[7]. The compression ratio (CR) is a metric used to evaluate memory compression efficiency[5], which is defined as follows:

$$CR = \frac{Compressed\ Program\ Size + Decoding\ Table\ Size}{Original\ Program\ Size}$$

Data compression can be divided into two types, called lossless compression, and the other is called lossy compression. Lossless compression is the use of compressed data, decompress, extract the data and the original data exactly the same. Lossless compression for the required data extracted raw data is fully consistent with the occasion[7]. Lossless compression divided into two type of technique, statistical based and dictionary based.

In statistical based technique, the compression process is done by the data is converted into a specific code one by one. The length of the output code varies, depending on the probability or frequency of the symbols that appear. In this method, a symbol with low frequency will be encoded with the number of bits more. The symbols with high frequency will be encoded with a fewer number of bits. Examples of compression algorithms with this method, such as Huffman Coding and Arithmetic coding.

Huffman Coding could need a large amount of memory due to the creation of an extended binary tree to generate a code for each symbol being processed from an alphabet[6]. The codes generated have variable length and can, under some

978-1-5090-5088-8/16/$31.00 ©2016 IEEE

circumstances, grow larger than the reserved memory of the application that is performing the coding[6].

II. DESIGN SYSTEM

In this study, the data logger has three main functions:

1. the data logger as a data storage device sensor measurement results,
2. the data logger as a device controller for other devices, and
3. the data logger as machine to machine (M2M) device for IoT

The design of the data logger system based on these functions, illustrated in the Fig.1.

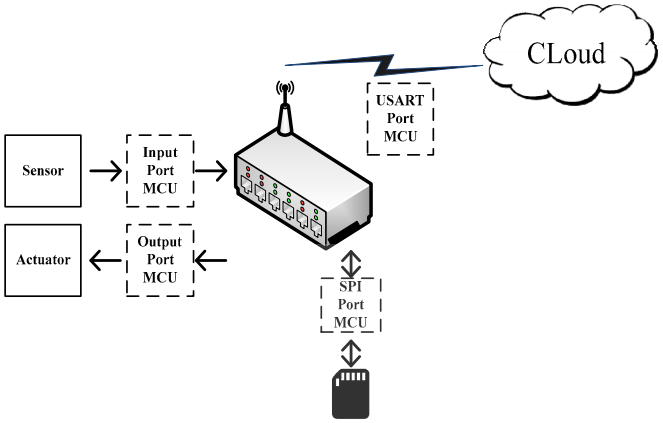

Fig. 1 Data logger design system

Sensors and actuators can be integrated on the device. This device will be used on hydroponic plant nutrition as a case study, with architecture design is shown in Fig. 2.

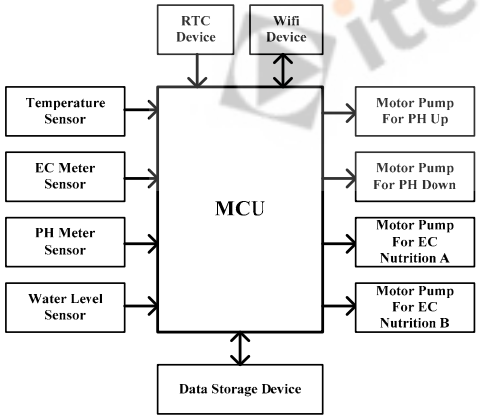

Fig. 2 Data logger Arcitecture

External data storage memory used is MMC with capacity 2GB type FAT16. Dictionary-based techniques used in the compression process, by replacing the data or symbols that have long codeword, into a specific code that has fewer codeword length. The codes are used in the encode process contained in encoding table.

Dictionary-based code compression techniques are popular because they provide both good compression ratio and fast decompression mechanism[16]. The basic idea is to take advantage of commonly occurring instruction sequences by using a dictionary. Dictionary technique is commonly used in embedded systems [15]. It can achieve an efficient CR, possess a relatively simple decoding hardware, and provide a higher decompression bandwidth than the code compression by applying lossless data compression methods[5].

On a regular computer system, a coding process can always request more memory to the operating system if it is available, but on an embedded system it may not have that possibility due to two main problems: first, an embedded system does not have a large memory, and related to the above, currently it is rare that an embedded system runs an operating system that manages the main memory[6]. It would be desirable that the embedded system used a minimum of main memory to perform the encoding process.

III. HARDWARE DESIGN

Microcontroller unit (MCU) as the main controller, used in the data logger. Based on Fig. 2, MCU integrated with real time clock (RTC), wireless transmission device, and data storage devices as the main component. sensors and actuators used for peripheral device. The sensors are integrated in the device consists of temperature sensors, electrical conductivity (EC) sensor, a potential of hydrogen (pH) sensor and a water level sensor. 4 pieces of water pumps for actuators. Schematic design of the data logger device, shown in Fig. 3.

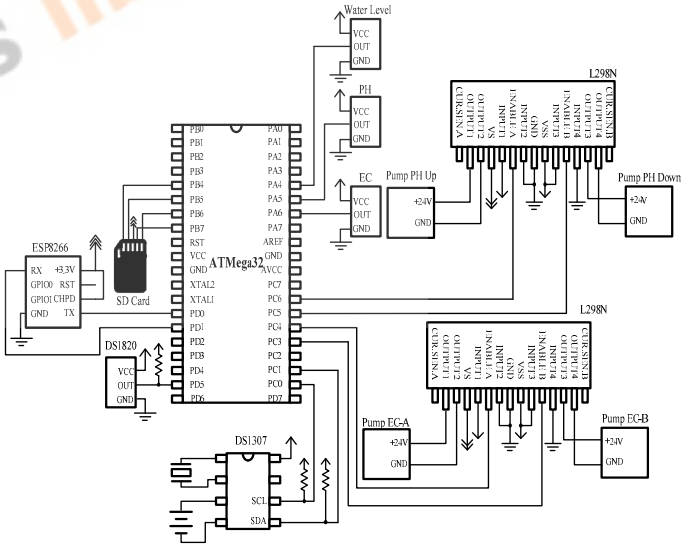

Fig. 3 Hardware design

ATMega32 as the main controller device has 32KB flash memory, 2KB SRAM and 1KB EEPROM. The power supply for this MCU between 4,5Volt-5,5Volt, and maximum external clock can be used is 16MHz. ATMega32 has 32 I/O port digital, 8 analog ports and there are multiple data protocols, including protocols SPI, USART, I2C, 1-Wire. This

MCU has a 2-channel 8-bit timers, one 16-bit timer channel and 4-channel PWM.

IV. SOFTWARE DESIGN

The data logger is designed for various processes. The first will sensing sensor, then next step to control process and transmits the data to the cloud. After the data is transmitted, then be compressed before being stored on the MMC.

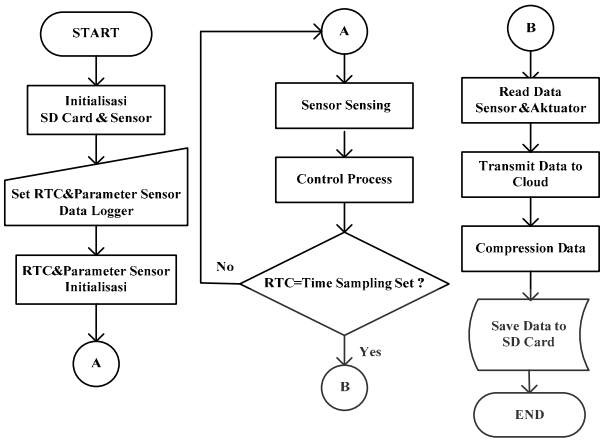

Fig. 4 Flowchart data logger process

In Fig. 4 data will be compressed, before being stored on the SD card. the compression process is done by using a dictionary. This compression technique is done by replacing the data of each character ASCII code into a specific code based on the data contained in the table encoding. ASCII constructed of 8 bits of binary data, so that the external data storage memory will allocate data storage by 1 byte to a character data type of text. Encoding table used in compression process, shown in Table 1.

TABLE I
ENCONDING TABLE

No	DATA		
	Before (Char)	Before (Hexa)	After
1	0	0x30	0x00
2	1	0x31	0x01
3	2	0x32	0x02
4	3	0x33	0x03
5	4	0x34	0x04
6	5	0x35	0x05
7	6	0x36	0x06
8	7	0x37	0x07
9	8	0x38	0x08
10	9	0x39	0x09
11	space	0x20	0x00
12	/	0x2F	0x0A
13	:	0x3A	0x0B
14	.	0x2E	0x0C
15	;	0x38	0x0D
16	Carriage Return	0x0D	0x0E
17	New Line	0x0A	0x0F

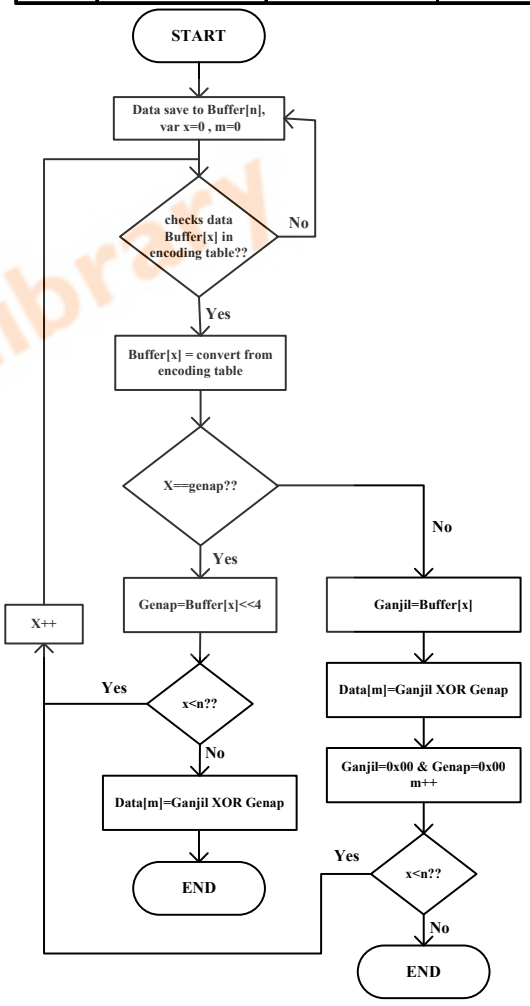

Fig. 5 Flowchart data compression process

V. EXPERIMENT

The testing process consists of two main parts, the first testing is processing time and the second testing is data storage on the MMC. The process of testing to be done by testing a time for sensing sensors process, control process, data compression process, data transmission and data storage process. The first test was performed 10 times in testing at each process. The calculation for time process by use timer 8 bit feature on the MCU. MCU timer is set for 10mS each time increment. The results of the calculation processing time is displayed in a format USART.

TABLE III
TIME PROCESS IN DATA LOGGER

No	Process	Time (ms)
1	*Average Sensing Sensor*	1353
2	*Average Control Fuzzy Logic*	52
3	*Average Compression Data*	54
4	*Average Transmisi Data to Cloud*	23
5	*Average Save Data to SD Card*	36
Total Process		1518

Table 2 shows the results of the first test, with total process is 1518 ms. Based on the test results are known the longest time on the sensor sensing process. Time on the sensor sensing are longest because the sensor used require considerable time in the process of measurement, so that the measurement results more accurate and precise.

The second test aimed to compare the amount of file data to be stored, before and after a compression. Tests conducted within 1 hour with time sampling every 5 seconds. The number of tests were conducted 10 times. The test results are shown in Table 3.

TABLE III
EXPERIMENT COMPRESSION DATA FOR 1 HOUR

Number of Test	File Size (Byte)		CR
	Before Compression	After Compression	
1	15840	7920	50%
2	16408	8204	50%
3	15932	7966	50%
4	16673	8337	50%
5	16752	8376	50%
6	16243	8122	50%
7	15987	7994	50%
8	17425	8713	50%
9	17660	8830	50%
10	17573	8787	50%
Average	16649	8325	50%

The size of the data results from the compression process is half of the original data.

VI. CONCLUSIONS

The require minimum time to process one cycle in data logger is ± 1518ms.

In 1 hour, SD card saves the data compression ± 8325 bytes when time sampling 5 seconds. So if the capacity of SD card is 2 GB or in the real size is 1887 MB, the SD card can saves measurement of data for ± 226667 hours. The duration of data logger to save the data, affected by the amount of data from the sensor or actuator to be stored in the device, and from configuration sampling time.

The rasio between compressed program size with original program size is 50%, that generated by this compression method. This method help us to addressing the data storage on memory that has small capacity, so it able to store two times more of the data.

REFERENCES

[1] H. Erdem, *Design and Implementation of Data Logger for Fuzzy Logic Controller*, Proc. of the IEEE Int. Conference on ICIT'02 on Industrial Technology, Vol.1, 2002, pp.199-204

[2] N. N. Mahzan, A. M. Omar, S. Z. Mohammad Noor, M. Z. Mohd Rodzi, Design of Data Logger with Multiple SD Cards, Proc. of the IEEE Int. Conference on Clean Energy and Technology (CEAT), 2013, pp. 175-180

[3] Oka Mahendra, Djohar Syamsi, *Design of a Dual-microcontroller Scheme to Overcome The Freeze Problem for a Smart Data Logger,* 2014 2nd Conference ICoICT, 2014, pp.314-319

[4] Glenis Moore, *Data Logger Modern Recorders*, IET Jurnals & Magazines, 1986, Vol.32, pp.132-136

[5] Wei Jhih Wang, Chang Hong Lin, *Code Compression for Embedded Systems Using Separated Dictionaries*, IEEE Transactions on VLSI Systems Journals, Vol.24, 2016, pp.266-275

[6] Marco Antonio Soto Hernandez, Oscar Alvarado-Nava, Francisco Javier Zaragoza Martinez, *Huffman Coding-Based Compression Unit for Embedded Systems*, Proc. of the IEEE Int. Conference on Reconfigurable Computing and FPGAs, 2010, pp.238-243

[7] Zhiyong Zhang, Xiaoning Li, Xiaofeng Li, *Study on lossless data compression based on embedded system*, Proc. of the IEEE Int. Conference on BIC-TA, 2010, pp.1225-1230.

[8] M. Kozuch, A. Wolfe, *Compression of embedded system programs*, IEEE International Conference Computer Design, 1994, pp.270-277

[9] R. L. Milidiu, E. S. Laber, and A. A. Pessoa, *A work efficient parallel algorithm for constructing huffman codes*, in Proc. Data Compression Conference (DCC '99), 1999, pp. 277–286.

[10] P. Berman, M. Karpinski, and Y. Nekrich, *Approximating huffman codes in parallel*, Electronic Colloquium on Computational Complexity (ECCC), 2002.

[11] S. Wong, S. Cotofana, and S. Vassiliadis, *General-purpose processor huffman encoding extension*, in Proc. of the Int. Conference on Information Technology: Coding and Computing (ITCC 2000), 2000, pp. 158–163.

[12] A. Jas, J. Ghosh-dastidar, M. eng Ng, and N. A. Touba, *An efficient test vector compression scheme using selective huffman coding*, *IEEE Trans. Computer-Aided Design Integr. Circuits Syst*, vol. 22, pp. 797–806, 2003.

[13] A. Moffat and J. Katajainen, *In-place calculation of minimum-redundancy codes*, in Lecture Notes in Computer Science. Springer-Verlag, 1995, pp. 393–402.

[14] A. Wolfe and A. Chanin, *Executing compressed programs on an embedded RISC architecture*, in *Proc. 25th Annu. Int. Symp. Microarchitecture*, Dec. 1992, pp. 81–91.

[15] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, *Improving code density using compression techniques*, in *Proc. 30th Annu. ACM/IEEE Int. Symp. MICRO*, Dec. 1997, pp. 194–203.

[16] Seok-Won Seong, Prabhat Mishra, *A Bitmask-Based Code Compression Technique for Embedded System,* in Journal IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol.27, 2008, pp.673-685