Middle-East Journal of Scientific Research 20 (9): 1051-1054, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.20.09.114108

Data Acquisition System Based on CAN Bus and ARM

M. Nirubama

Department of ETC,
Bharath University, Chennai-73, India

Abstract: The application development of data acquisition system developed on the platform of 32 bit RISC ARM microprocessor technique, embedded software technology, embedded User Interface technology, CAN bus communication technology, information storage and management technology etc. The embedded platform designs on lpc2129 core and Real Time Operating System is embedded into the platform. The development trend and project focus of control system is networked control. The Field bus Control System (FCS) which is important segment in networked system is widely applied in control system. The CAN (Controller Area Network) bus which is strong fault tolerance, high reliability and low cost played an important role in FCS. However, the host computer of control system which adopts CAN bus is industrial computer. This computer usually costs much and needs more space. If the embedded equipment is applied in control system, the cost is very low and the control system has advantage in easy customization and extension. The Real Time Operating System has control over the system. In this project the freeRTOS is used as RTOS, A typical use of the FreeRTOS kernel is when one needs to have some kind of (time critical) control algorithm while also providing user-control and sensor monitoring. The system has three tasks to control Temp., Pressure and LCD. In this system the Temp. task has highest priority then Pressure and at last LCD task. The RTOS give dynamic control over the Task priority rather than the static control in CAN and it also give the advantage of RTOS environment.

Key words: CAN bus communication · Field bus Control System · RTOS environment

INTRODUCTION

The project focus on the modern industrial data acquisition system, it provides the real time data access from sensor and continuously showing this data on LCD. The Real Time Operating System has control over the system. In this system we have three tasks Temperature (Temp.), pressure & LCD task. The Temp. task has highest priority after this Pressure and LCD task has least priority. The RTOS schedule the tasks according to their priority. In this project the freeRTOS have used, because this is ideal for time critical small embedded system [1].

The data is collected from the sensor and transmit to the ARM processor then, ARM transmits these data to CAN controller. The CAN controller establish the CAN protocol via transceiver between two node. After receiving the data from first node we are shows this data on LCD [2].

Data Collecting Node: The data collecting node have lpc2129 ARM TDMI microcontroller it has inbuilt two CAN controller, one Temp. Sensor LM35 and capacitive pressure sensor. The sensors convert the physical property into electrical signal. LM35 is precision Temp. sensor which output voltage is directly proportional to the Temp. in degree centigrade. Output of the capacitive pressure sensor depends on the charge storage between two plates [3].

The sensor output is analog signal and microcontroller works on digital signal, so we have to convert it in digital signal. ARM controller has built in ADC therefore we give the sensor output to ADC of the ARM microcontroller. The digitally converted signals transmit to CAN controller which implement the CAN protocol. The mcp2551 transceiver is used to interface the node on CAN bus [4].

Data Display Node: The Data Display Node consists of lpc2129 ARM TDMI microcontroller and Samsung 162A LCD. The mcp2551 transceiver receive data from CAN bus and forward to the CAN controller, the CAN controller disconsolate the frame and transfer data to LCD. The LCD continuous shows the required data [5].

Corresponding Author: M. Nirubama, Department of ETC, Bharath University, Chennai-73, India.

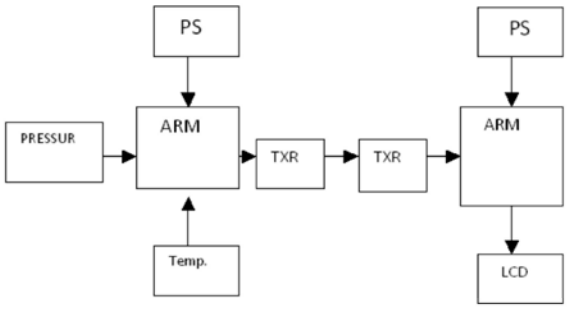

Fig. 1: System Block Diagram.

Hardware Description: This design consists of the collection and display of data which are independent. The data collection module receives data through CAN bus and the data display module display these data via LCD by Embedded in ARM.

The LPC2129 are based on a 16/32 bit ARM7TDMI-STM CPU with real-time emulation and embedded trace support, together with 128/256 kilobytes(kB) of embedded high speed flash memory. A 128-bit wide internal memory interface and a unique accelerator architecture enable 32-bit code execution at maximum clock rate. For critical code size applications, the alternative 16-bit Thumb Mode reduces code by more than 30% with minimal performance penalty [6].

With their compact 64 pin packages, low power consumption, various 32-bit timers, combination of 4-channel 10-bit ADC and 2 advanced CAN channels or 8-channel 10-bit ADC and 2 advanced CAN channels (64 pin packages) and up to 9 external interrupt pins these microcontrollers are particularly suitable for industrial control, medical systems, access control and point-of-sale [7].

Number of available GPIOs goes up to 46 in 64 pin package. Being equipped wide range of serial communications interfaces, they are also very well suited for communication gateways, protocol converters and embedded soft modems as well as many other general-purpose applications [8].

The LCP2129 has inbuilt CAN Controller, so for interfacing CAN bus & controller we have to use external CAN transceiver. The MCP2551 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2551provides differential transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s.

The MCP2551 CAN outputs will drive a minimum load of 45Ω, allowing a maximum of 112 nodes to be connected (given a minimum differential input resistance of 20 kΩ and a nominal termination resistor value of 120Ω).

The RXD output pin reflects the differential bus voltage between CANH and CANL. The low and high states of the RXD output pin correspond to the dominant and recessive states of the CAN bus, respectively.

The Display component consists of LPC2129 ARM TDMI microcontroller and Samsung 162A LCD.

CAN Bus System: The CAN is a serial communications protocol which efficiently supports distributed real-time control with a very high level of security.

Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bit rates up to 1M bit/s. The intention of this specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has different aspects regarding e.g. electrical features and the interpretation of data to be transferred. To achieve design transparency and implementation flexibility CAN has been subdivided into different layers.

Principles of Data Exchange: CAN is based on the "broadcast communication mechanism", which is based on a message-oriented transmission protocol. It defines message contents rather than stations and station addresses. Every message has a message identifier, which is unique within the whole network since it defines content and also the priority of the message. This is important when several stations compete for bus access (bus arbitration).

As a result of the content-oriented addressing scheme a high degree of system and configuration flexibility is achieved. It is easy to add stations to an existing CAN network without making any hardware or software modifications to the present stations as long as the new stations are purely receivers. This allows for a modular concept and also permits the reception of multiple data and the synchronization of distributed processes. Also, data transmission is not based on the availability of specific types of stations, which allows simple servicing and upgrading of the network.

Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus 4cabling (differential output). It also provides a buffer

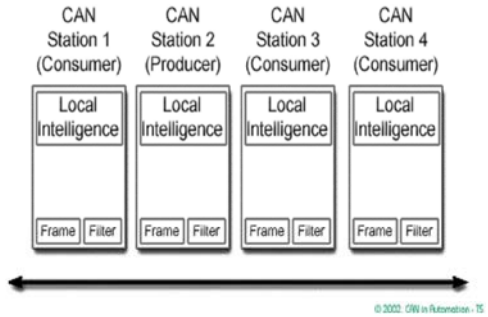

Fig. 2: CAN Consumer & Producer station.

between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outsider sources (EMI, ESD, electrical transients, etc.).

The CAN bus has two states: Dominant and Recessive. A dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g.1.2V). A recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). The dominant and recessive states correspond to the low and high state of the TXD input pin, respectively. However, a dominant state initiated by another CAN node will override a recessive state on the CAN bus.

Free RTOS: FreeRTOS is a free real-time operating system kernel. It is aimed at small embedded real-time systems. Since most of the code is written in the c programming language, it is highly portable and has been ported to many different platforms. Its strength is its small size, making it possible to run where most (or all) other real-time operating systems won't fit.

Free RTOS Scheduler: FreeRTOS uses a highest priority first scheduler. That means that the task with highest priority is always run. This is achieved by having a preemptive scheduler that at a tick-interrupt decides if the current task is allowed to continue executing or if it needs to be switched for another task. To be able to do this FreeRTOS demands that every task is assigned with a priority. The scheduler then uses this priority to schedule the task with highest priority. Tasks that have the same priority is given "fair" process time by round robin. We can see the advantages of having such a simple and small scheduler for a small embedded system. Still one loses the possibility to achieve hard real-time by not having any kind of deadline based scheduling. The developer is given the choice of having a preemptive or a cooperative scheduler. In preemptive mode a task can be preempted unlike in cooperative mode where it's up to all tasks to give away the CPU "often" enough so higher priority tasks get to run.

A typical use of the RTOS kernel is when one needs to have some kind of (time critical) control algorithm while also providing user-control and sensor monitoring. Propose we have a system with some sensor that needs an amount of processing which leads to a control signal (say a water flow). It's critical that we measure at controlled timings to make sure that there isn't an over flow. This will be our highest priority process. We then have an LCD-display which must be updated with the current flow and how much to put in each bottle. Also we might have a keypad that needs to be read so we can configure the system in some way. We now have three tasks that need to be executed with different priority.

The RTOS solution to this problem might be an AVR with three different tasks each of them given a specific priority. Then we schedule the tasks to run at different intervals. The preemptive scheduler of RTOS makes sure that the most important task (the control loop) is always run at time. We now have a small system that performance this task.

RESULTS

The project is working successfully and data is display on LCD continuously. The RTOS schedule all tasks according to their priority. In this project the three tasks have used, first is temperature task second is pressure task and last is LCD.

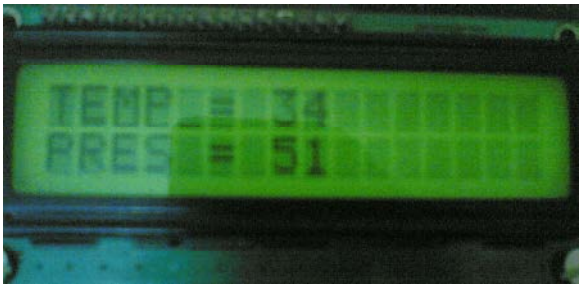

Fig. 3: LCD output result.

The Tasks Priorities Are in Following Order:

- Temperature task.
- Pressure task.
- LCD task.

The RTOS always give preference to the temp. task because it has highest priority, in other word whatever the tasks RTOS schedule at any time if temp. task comes RTOS stopping the current task and give the preference to temp. task. After completing the temp. task RTOS will schedule the remaining task(in other word the current task running before the temp. task). Then RTOS schedule the next task which has highest priority in the scheduling table.

The output is display by LCD as shown on above Figure 3. The LCD continuously shows the current temperature and pressure and updating the data on regular basic of interval as per predefine.

The data is efficiently transmitted between two node by using CAN bus. The CAN bus provide the error checking mechanism for the data frame. The CAN bus also provide the imminence from electromagnetic interference due to the differential voltage CAN_L and CAN_H respectively. This feature is good in industrial environment where the electromagnetic interference is high.

CONCLUSIONS

The data terminal based on ARM embedded RTOS system is actualized. The monitor has advantage in easy customization and extension. Through the experiment, it is proved that the veracity of parameter transmit based on CAN bus is very high and the display module can easily plug and play. Therefore, this control method and communication module has more reliability and mobility. As a field equipment bus, the CAN is more reliable and higher performance to price ratio than other bus.

To sum up, because of the superior performance of ARM RTOS system and the reliability and real-time performance of the data transmitted on CAN bus, the CAN bus combined with ARM-Linux is appropriate to industry and it has wide applied foreground in process control, motor manufacture, agricultural equipment, iatrical equipment and so on.

REFERENCES

1. Peng Yang, Le Yang and Xin Guo, 2009. Research on Embedded Data Display Unit Based on CAN Bus, proceeding IEEE Conference on Industrial Electronics and Applications, pp: 3498-3501.
2. Lee, K.C. and H.H. Lee, 2004. Network-based Fire-Detection System via Controller Area Network for Smart Home Automation, IEEE Transactions on Consumer Electronics, 50(4).
3. Guest Editorial Special Section on Communication in Automation-Part I, IEEE Transactions on Industrial Informatics
4. Guest Editorial Special Section on Communication in Automation-Part II, IEEE Transactions on Industrial Informatics
5. Ramanarayana Reddy, S., 2006. Selection of RTOS for an Efficient Design of Embedded Systems, IJCSNS International Journal of Computer Science and Network Security, 6(6).
6. Rafael, V., Aroca1 and Glauco Caurin1, A Real Time Operating Systems (RTOS) Comparison, Nuclear Science, IEEE Transactions.
7. Sohal, V., 2001. How to really measure real-time System, In Embedded System Conference.
8. Karl Anderson and Robert Anderson, A comparison between FreeRTOS and RTLinux in embedded real-time systems, In Embedded System Conference.