

Lab 3g Distributed Data Acquisition

This laboratory assignment accompanies the book, *Embedded Microcomputer Systems: Real Time Interfacing*, 2nd Edition, by Jonathan W. Valvano, published by Thomson-Engineering, copyright © 2006.

This lab must be performed in teams of 4 to 6 students. Approval of the team by the TA is required before the preparation is started. Existing lab groups can be merged together to form the team (no splitting of existing groups). Only one version of the software need be developed for each team, but the number of operational nodes in the network is equal to the number of students on the team minus one. For example a group of 6, should create a network with 5 nodes.

- Goals**
- Develop a layered communication system,
 - Design and implement a hardware/software interconnect protocol.
 - Use communication skills to work effectively as team.

- Review**
- Valvano, 2nd Edition, Section 14.3 on CAN networks
 - Chapter 11 on the CAN in the 9S12 data sheet **MC9S12C128_V1.pdf**
 - MCP2551 data sheet **MCP2551.pdf**

- Starter files** • **TxFifo RxFifo** from **SCIA**

Background

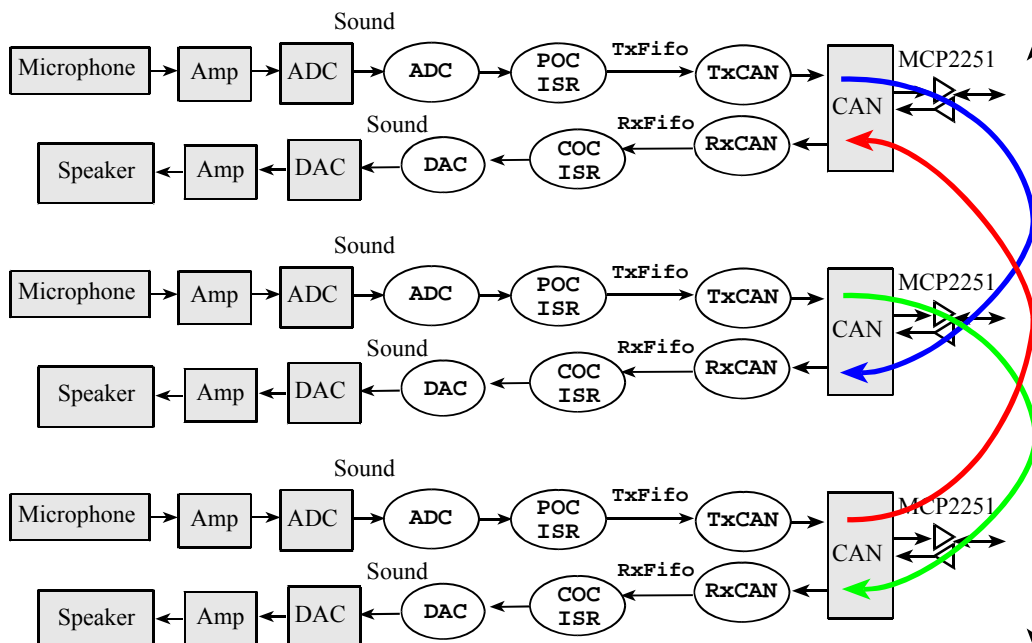


Figure 3.1. Data flow graph of the distributed data acquisition system.

The objective of this lab is to design a distributed audio monitoring system using a CAN network. Each microcontroller node will measure one sound channel using a microphone (from Lab 2). Each node will also have one sound play device (Lab 2), and a CAN network interface. Figure 3.1 shows the data flow graph of the distributed system. The sound recording channel (ADC) of one node will be uniquely connected to the sound play channel (DAC) of one other node across the network. For example, assume there are three nodes numbered 1, 2, and 3. In this system, the sound recorded on microphone 1 plays on speaker 2, the sound recorded on microphone 2 plays on speaker 3, and the sound recorded on microphone 3 plays on speaker 1. You can implement whatever sampling rate you wish.

There are four background threads on each node. The producer thread (labeled **POC** in Figure 3.1) will periodically measure sound and send its local measurement to the other nodes on the network. The **TxCAN** thread is also an interrupt service (may be implemented as a periodic interrupt or as a flag-trigger interrupt) and is responsible for transmitting messages. The CAN identifier will specify the connection, For example, in a network with three nodes, there are three connections (i.e., 1-2, 2-3, or 3-1). The data field can be formatted however you wish. Both the data acquisition (**POC**) and network transmission (**TxCAN**) will be performed in the background. The **RxCAN** thread, also running in the background, will accept messages from the other nodes. You can filter the messages in hardware or in software. The appropriate data will be transferred to the play device, which is implemented as a periodic background thread, **COC**. The primary responsibility of the **main** program is to initialize the system, check for errors, assist in debugging, and restart the system if appropriate. You should avoid using SCI input/output so that the Metrowerks debugger can be used. As always, you are allowed to implement the system in an alternate manner, as long as the basic educational objectives of the lab are achieved. In particular, the specifications include

- Each node measures sound locally in real time using its microphone.
- Each node plays sound on its speaker that was measured at a different node.
- There are at least $n-1$ nodes, where $4 \leq n \leq 6$ is the size of the group.
- The CAN interface is used.
- Interrupt synchronization is used in an appropriate manner.
- The hardware/software system is modular.
- The system uses FIFO queues.
- You run the system at various sampling rates in order to determine the maximum bandwidth.

Figure 3.2 is a possible call-graph of the system, showing the modular structure of the hardware/software systems. The output compare hardware can be used to implement real-time input and output. First-in first-out queues are used to pass data between threads. The **ADC** **DAC** modules are device drivers implementing the analog to digital and digital to analog conversions respectively. The **CAN** network driver includes a transmit channel (via **TxFifo**) and a receive channel (via **RxFifo**). You are free to use the CAN interrupts themselves or to implement background I/O using periodic polling with a periodic interrupt.

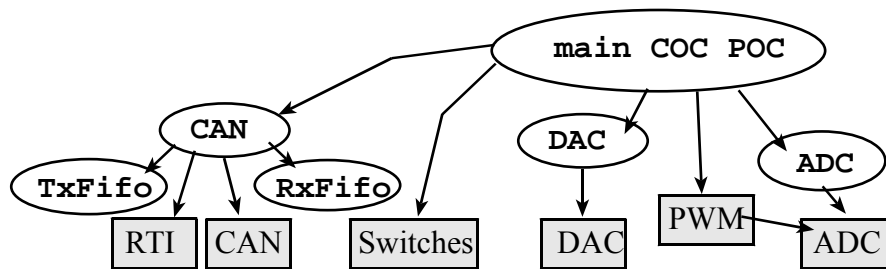


Figure 3.2. Call-graph of the distributed data acquisition system.

Choose a CAN channel bandwidth (e.g., 250,000 bps) and leave it as a constant. To change data bandwidth, you will adjust the ADC and DAC sampling rate. Start with a data acquisition/transmission rate so slow (e.g., 500/sec) that collisions will be rare, and all the FIFOs are usually empty. Then, keeping the CAN channel bandwidth fixed, increase the sampling rate (number of ADC samples/sec, number of CAN transmissions/sec and the number of DAC outputs/second) until the maximum bandwidth is reached, without loss of data. For example, if there are 5 nodes in the network, each node is transmitting at 500 samples/sec, each sample represents 2 bytes of information, and no data is lost, then the network bandwidth will be 5000 bytes/sec. It is possible to implement this network using just the **Data Frame**, ignoring the **Remote Frame**, the **Error Frame**, and the **Overload Frame**. Similarly, you can implement just the **Standard CAN 2.0A**, instead of the **Extended CAN 2.0B**. CAN does force a priority structure on the network, but you can implement priority however you wish. You could enable the CAN interface to accept all messages. However, it will be more efficient to implement CAN filters at the hardware level.

Preparation (do this before your lab period)

1) Design the hardware-level communication interconnection. Draw the circuit diagram using a CAD program like **ExpressSCH**, labeling resistors and pin numbers. The MCP2551 CAN interface circuit will be used. You should add a bypass capacitor for the MCP2551. Leave pin 5, V_{REF} , not connected (mistake in book Figure 14.8). A resistor

from pin 8, R_s , to ground can be added to restrict the slew rate on the channel. Limiting the slew rate will reduce EMI emissions. In this lab, however, you can ground pin 8. You will need to construct a 3-wire cable and terminating resistors in order to physically build the network (you can actually build the network during lab period, but collect the materials as part of the preparation). Each node will require three connections CANH, CANL and ground.

2) You will need two FIFO modules, one for the CAN transmitter and one for the CAN receiver. You might be tempted to simply copy and paste the **TxFifo** and **RxFifo** from the SCIA project without modification. The **TxFifo** and **RxFifo** from the SCIA project are appropriate when transferring individual bytes between threads. On the other hand, I strongly suggest you design your own FIFO, so that entire messages (node number and sound) are entered and retrieved at one time. Assume for example, each message is 3 bytes; now consider the mess you will be in if you successfully put one byte into a FIFO, but the FIFO is full when you try to put in the second byte. You should write a separate main program that you will use to verify the correctness of your FIFO modules. The size of these FIFOs will be adjusted later during the network-testing phase of the project, as you try to optimize bandwidth given the limited resources (CAN channel bandwidth, 9S12 RAM, and 9S12 execution speed).

3) Write the low-level CAN I/O device driver. Both the CAN transmitting and CAN receiving should occur in the background. Be careful to consider what your software does if

TxFifo is full on a **TxFifo_Put**

TxFifo is empty on a **TxFifo_Get**

RxFifo is full on an **RxFifo_Put**

RxFifo is empty on an **RxFifo_Get**

All four of these conditions might occur while running in the background with the **I** bit set. If the **I** bit is set, YOU CAN NOT LOOP waiting for the FIFO condition to reverse. For example, look at the interrupt-driven SCI device driver (SCIA project). In particular, see what the RDRF ISR does if it receives an incoming character, but the call to **RxFifo_Put** returns a full error. In addition, see what the TDRE ISR does if the transmitter is idle, but the call to **TxFifo_Get** returns an empty error. You should write two main programs, one that transmits only and the other that receives only. These programs will be used to debug the network channel. Again, if you avoid using SCI I/O, you can use the Metrowerks debugger. For example, rather than printing messages on the SCI output, dump data into global memory and observe it using the debugger.

4) Finally, write the main program that accepts sounds from the microphone, and plays remote data from the network. The main program will be responsible for error checking. You could add a LCD display showing debugging information and/or performance information (network bandwidth). You can count the number of packets lost by sending known sound sequences (like ramp up ramp down). You can also see a lost packet when the ISR tries to transmit a packet, which calls **TxFifo_Put** and the **TxFifo** is full. Similarly, a packet will be lost if the CAN receiving ISR calls **RxFifo_Put** and the **RxFifo** is full.

Procedure (do this during your lab period)

1) The first step is to build the physical network. There should be terminating resistors at each end of the cable to eliminate reflections and to limit the rise/fall times of the voltages on the network.

2) The second step is to debug your two FIFO modules.

3) Next you should debug the CAN transmitter. Run the main program that just transmits and observe the signals on both sides of the transmitting MCP2551 and a receiving MCP2551. Compare these signals to the data sheets of the 9S12 CAN and the MCP2551 interface chip. If you are not transmitting frames properly, then the scope measurements should isolate the problem as hardware or software.

4) The next step is to run one node in transmit-only mode and a second node in receive-only mode. If you are not receiving properly, then you can measure the input signals on PTM of the receiver to verify the frames are being properly transmitted. Next, you can test with multiple receive-only nodes, and multiple transmit-only nodes. Once these tests are successful, then you can integrate both transmitting and receiving into each node, and test it again.

5) Once the CAN I/O device driver is successfully debugged, then it should be integrated into the sound recording and play back modules.

6) Measure the maximum sustained bandwidth of the system, under the conditions that virtually no data is lost. Keep the CAN channel speed fixed at 250,000 bps and increase the sampling rate, while measuring the number of lost packets. "Sustained" means the measurement should occur over many seconds, so that the steady state behavior is measured, and not a start-up behavior measured when all the FIFOs are initially empty. Which factor limits the bandwidth? Consider factors such as the speed of the CAN channel, the speed of the ADC converter, execution speeds of various background threads, execution speed of foreground program, or the LCD display speed.

Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum)

B) Hardware Design

Detailed circuit diagram of the physical layer of the network (preparation 1)

C) Software Design (no software printout in the report)

Define the communication protocol in the comments of the appropriate header files

Draw figures illustrating the major data structures used

D) Measurement Data

Draw CAN signals measured on both sides of the transmitting MCP2551 and a receiving MCP2551.

Measure the network bandwidth

E) Analysis and Discussion (1 page maximum)

Document major features of the network

Discuss the factors limiting bandwidth of the network

F) Post-mortem concerning team member interactions (attached to the report)

1) Each team member evaluates each other team member including oneself

Simply list one or two weaknesses.

Simply list two or three strength characteristics.

2) Major failures in the way the team interacted (if any)

3) Major successes in the way the team interacted

G) Peer Review (each student submits independently and confidentially directly to the TA)

Classify each team member including oneself as:

- worked harder than average (explain), worked an average amount, worked less than average (explain)

Partial Checkout (show this to the TA after the first week)

No printed documents are required, but you should demonstrate simple CAN messages being sent from one transmitter to one receiver, including scope measurements of the CAN transmissions. There will be a 10-point grade reduction on this lab, if your group is unable to complete this partial checkout.

Checkout (show this to the TA)

You should demonstrate the major features of your communication system to the TA, including your measurement of bandwidth and your method to detect lost packets.

Your software files will be copied by the TA during checkout.