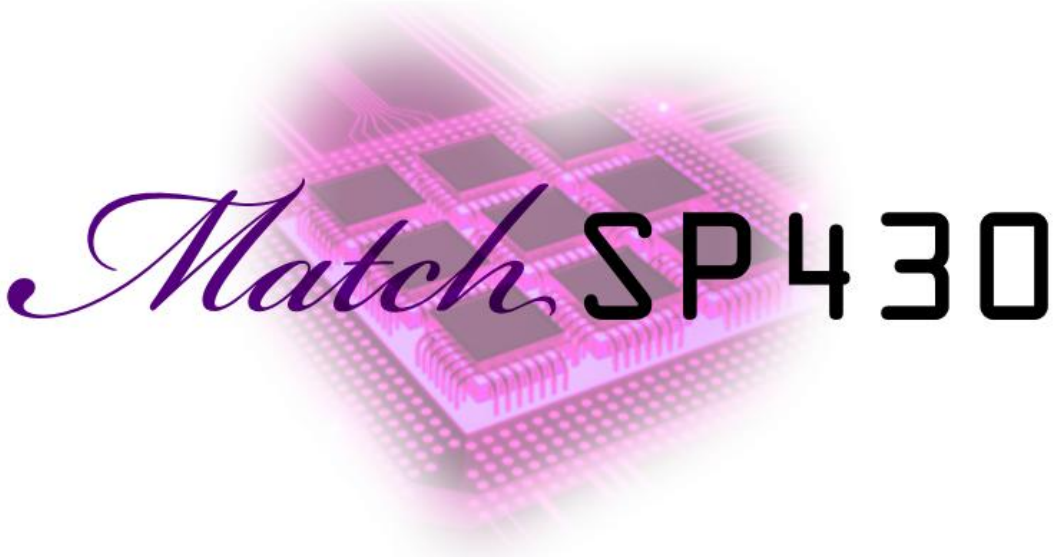
Brian Ritchken
bjr96

Brendon Jackson
btj28



Match me!

MatchSP430 is the ultimate matchmaking utility for 16-bit mixed-signal microcontrollers. It allows the user to input a profile (which consists of **first** and **last** name, **bio**, **gender**, and who you are **interested in**. Using the on-board radio, MatchSP430 then seeks out other users with compatible profiles and displays them to you, forming a list of connections. From there, both people have the option to match with the other, and if both agree, a match is made. This is all done through a beautiful terminal-style user interface using PuTTy and UART.

Under the (micro) surface

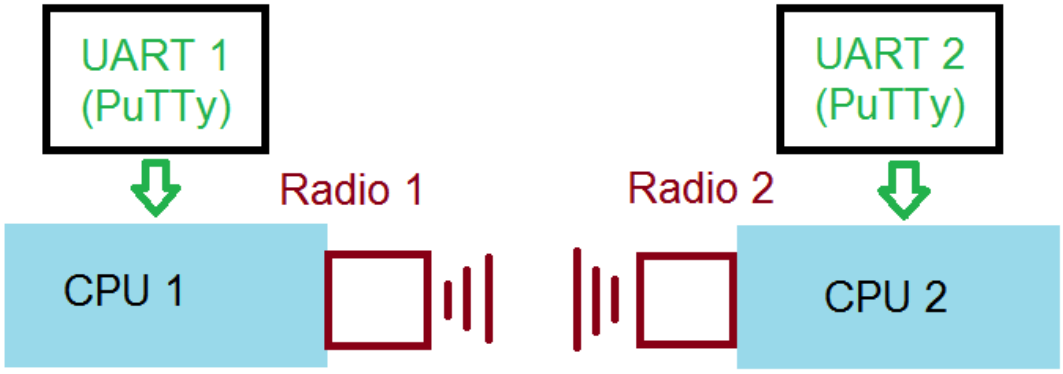


Figure 1: A block diagram detailing interaction between two users.

In order to implement MatchSP430, we needed to transmit and manage a lot of information. Each processor needed its own unique ID, along with an entire profile. Moreover, in order to support multiple connections and matches, the processors needed support to scan for nearby processor and keep track of their data. Therefore, we had to be careful that we didn't waste too much memory.

Other than the code for managing the radio (which is heavily commented) and UART (again, commented with helper functions), we had to implement several data structures. The basic functionality of the processor involves two key concepts: packets and queues. Packets are sent via the radio, and contain information about profiles or messages. They hold all the matchmaking data and transfer it accordingly into queues, which contain the data themselves for the user.

Software

Our code uses several high-level functions that interface between users. These include:

`main` simply initializes the UART (with a call to uart_init) and radio for communication. It sets several variables like the ID of each user. It ends by calling setupRoutine.c, which is described below.

`setupRoutine` helps the user to input a profile whenever the device is restarted. It loops 8 times, prompting the user to enter information as it walks through the help menu of MatchSP430.

`commandEntered` checks for user-inputted commands, like **#search** or **#self**. The command **#help** displays a description of all the commands to help the user.

`UpdateMessQ` handles the input from uart, adding it to either the queues or a packet to be sent via the radio.

`innercommands` handles any "applications" that start to run after a command is called like **#message** or **#matchwith**.

`bloop and receivedBloop` use the radio to communicate packages. One is called by the sender, the other by the receiver.

Packets

In order to send any information via the radio, we first assemble the data into a packet. This packet contains:

1 byte "Size"	4 byte "SENDER"	4 byte "RECEIVER"	1 byte "PACKET_TYPE"	N byte "Data"

Figure 2: Visual representation of a packet we would send

While much of this packet is the same as the provided code, we added the specific byte "PACKET_TYPE" which is used to interpret the data. There are 7 types of packets detailed below.

First Name (Encoded as 0x01): The first name of a user.

Last Name (Encoded as 0x02): The last name of a user.

Bio(0x03): A short description of a user.

Gender(0x04): The gender of the user.

Interested In(0x05): The gender the user is interested in.

Message(0x10): A message to be sent to another user.

Match: A packet with no data that strictly uses the SENDER data to transmit match information.

Queues

Just as implemented in other labs, we required the use of two queues to keep track of user data. These were implemented with linked lists, for which we have several helper functions. For each processor, connections (found by scanning for recipients through all the frequencies we used [in our case, 192.168.41.0 through 192.168.41.10]) are automatically stored in the connections queue. If matches are requested by both matched parties, then these connections are removed from the connected queue and added to the matched queue.

Testing & Challenges

The majority of our time was spent testing, and we ran into a huge number of bugs just trying to configure the radio and UART. Each was very finicky, and one wrong line of code could lead to unpredictable outcomes. Even encoding values into these devices proved painful and time-consuming.

This project really challenged our knowledge of data types, as we frequently had to convert from integers to strings and characters, and use pointers to save memory.

We tested with two laptops, although it would have been nice to have a third easily-programmable computer nearby to act as a third or fourth profile. However, because of the way our data is stored, it was easy to add users and testing multiple users proved somewhat successful.

All in all, MatchSP430 was a huge success! Searching, user profiles, and online chat works beautifully and consistently. Matching works as well, although for demonstration we showed the chat between normal connections (i.e. unmatched). Finally, we would have liked to have ASCII images for the profiles, and while we drew up a few (of Professor Martinez, for instance), they proved too hard to display properly.

References

In order to initialize the radio code, we used the radio drivers and code that was provided on CMS. Moreso, we used previous labs as references (including the given code there) and a few initialization instructions from the TI support threads online.