ECSE426 Microprocessor Systems - Fall 2009

Final Project: A Wireless Game of Bulls and Cows
Partial Demonstration: Nov 12/13
Final Demonstration: Nov 25/26
Lab Notes (final submission): Dec 3

Project Overview

This final project will provide you with the opportunity to build a self-contained embedded system that will allow a user to play a game of Bulls and Cows against another user via a wireless link. This final project includes many elements that you will later find in your engineering career, namely a complex project, deadlines to meet and a team of engineers that must reach a common goal. The project has specifications which are purposely incomplete and include many areas where team decisions have to be taken and assumed. Typical engineering projects often start as an abstract solution to a problem or as a novel idea in need of a physical implementation. It is the duty of the engineering team to turn this vision into reality and apply the many trade-offs which ultimately lead to a good (or bad) product. The Apple iPod, for example started out as a MP3 player, like many others on the market. However, subtle differences in implementation, user interface and a good marketing turned it into an industrial success story.

Usage Scenario

Bob and Alice really like the pen and paper game of code guessing called Bulls and Cows and think that there is a market for a small device that will let two players play one against another via a wireless interface. They envision that in a room, many people would have their Bulls and Cows gaming unit (they called it BCGU) and could play games in a distributed manner. In a typical scenario, someone in the room (Bob) could be bored and turned on his BCGU, selecting the game host mode. His BCGU would start announcing that a game host is available via the wireless interface built into the product. Alice would be someone else in the room and also bored, so she would whip out her BCGU and check if anyone in the room is hosting a game. Her BCGU would detect Bobs game and she can connect to do a code cracking session against Bob. When a game begins, each unit would let the players enter 4 digit secret code and the remote unit will return the number of good digits in the proper position (Bulls) and the number of digits that are present in the remote secret code, but not in the right position (Cows). Each player attempts to guess the other player's secret code. The unit would need to make a good effort at keeping the game active even in the presence of interference from other radio transceivers in the ISM band. It its initial version, the BCGU will automatically return the number of Bull and Cows resulting from the other player attempt at guessing the code. The game would keep track of the number of guesses and each player could view its last 4 moves on the screen. Bob and Alice decided to open their game protocol such that many vendors supporting the wireless standard could implement the game on their platform.

Game Requirements and Project Goals

A good overview of the pen and paper version of the game can be found here:

http://en.wikipedia.org/wiki/Bulls_and_cows

Many elements are needed to make this product possible. The first is the ability to make a self-contained unit that will host a screen (basic requirement: 2x40 character-based LCD display), an input mechanism (keypad) and a wireless transceiver. Furthermore, many radio parameters are possible with the CC2500, so some method of permanent storage of the transceiver configuration is needed. This way, the device parameters can be recorded once and the unit will power up with the proper settings (channel/frequency, output power level, etc.).

Another important aspect of the gaming platform is to built a standardized protocol that will allow different "flavors" of the gaming platform to play one against another independently of their exact implementation details. This protocol must define the method how the gaming units find each other, how they can initiate a game and handle network error conditions.

Unlike the official version, this one involves more speed and less brain power, since the two gaming unit do not play turn by turn, but rather involve playing relatively quickly to guess the other gamer's secret code (this is done to simplify the radio protocol and reduce the number of corner cases which could result in deadlocks).

The requirements will thus be to produce a gaming unit that can let users play a game of Bulls and Cows against another unit while respecting the radio protocol given below.

Protocol Description

In order for games to be played via the wireless link across teams, a single well-defined protocol be need to be used for the communication.

The protocol will attempt to define the following elements:

- Modulation type and data rate
- Packet format
- Game host discovery
- Game start negotiation
- In-game control/status messages
- Proprietary extensions

It is possible that along with the project development, some teams find holes in the protocol specifications. Those issues will be addressed through the WebCT discussion board, so teams should monitor the boards for protocol tweaks and updates if needed.

Modulation type and Data Rate

To ensure compatibility between gaming units , the following radio configuration should be used : 256kBps data rate, MSK modulation, 540 kHz digital filter bandwidth. The operating frequency is configurable, but the TAs will assign frequency values for each team during development to avoid getting too much interferences. Be sure to plan for a configuration table for the radio settings. You should not need to re-compile your code to account for a frequency change. Just changing a value in the Flash memory and a reset of the unit should allow modification in radio parameters.

Note: To avoid any issues related to the CC2500 errata, you should use the PKTCTRL.CC2400_EN =

1 mode. In this mode, data whitening and CRC_AUTOFLUSH cannot be used. To simplify the coding and design of the software and limit the memory usage, packets are limited to a total size of 64 bytes or less.

General packet format

Length(1 byte); destination address(1 byte); source address(1 byte); Packet Type Identifier(1 byte) ; Packet body(variable length).
* note: the semicolons are field separators, do not send ";" in your packet.

Host Beacon

The gaming host unit must send a periodic identifier indicating that it is ready to host a game and this identifier must tell the receiver how to contact this gaming host. The interval between beacon identifiers should be constant, but randomly selected between 1 and 2 seconds. The reason for the interval variation between hosts is that if all hosts had the same beacon interval, and two hosts happened to be sending their beacon at the exact same time, they would clobber each other's beacon forever. The beacon is sent with the broadcast address, such that all listeners will be able to pick up the packet.

Beacon format:
LEN(1);DST(1, broadcast address);SRC(1);'H';'Player name'(up to 20 bytes, null terminated string)

Client Connection Request

The gaming unit can also be used as a client. The user simply selects this option via its user interface. As a gaming client, the unit will enable its reception on the configured channel and wait for beacons. The user should be able to exit this mode if it finds that his unit is unable to connect to a gaming host. However, if a beacon is heard, the client will wait between 0.1 and 0.4 seconds (random delay) and request a game from the host. It does so by replying with a packet that contains the host direct address (contained in the beacon) and provide this host with its own address as part of the connection request. If the host receives and accepts the connection request, it will reply with a connection granted message and the gaming mode can begin. The client will know that the host accepted the game request because heartbeat packets (see below) will begin to be received.

Connection Request Format:
LEN(1);DST(1);SRC(1);'R';'Player name'(up to 20 bytes, null terminated string)

In game Packets

Heartbeat Packets

Once established, the host and client do not really know if they can talk to each other until they send a packet to one another. Therefore, it is important that some kind of connection status be exchanged regularly to know if the host/client link is useable. Because the client or host send guesses to each other after a short (or long) reflection of the user playing the game, another mechanism must be put in place to constantly monitor the link. This is the concept of heartbeat mechanism. Once a game is established, the host and client will exchange a packet every 1-2 seconds (random interval). Each end will monitor those heartbeat packet. If after sent 20 heartbeats either end of the link has not received a

single heartbeat packet, both end will independently terminate the game and indicate to the user that the connection has been lost.

Heartbeat packet format:

LEN(1);DST(1);SRC(1);'Z';'<NUM>'(1 byte indicating how many heartbeat I sent since I last got an heartbeat)

Guess Attempts

Once a game has been established, each player can attempt to guess the other player's code. We could have decided to make it turn based (you can implement this as a software option). However, the basic gaming mode is that each unit can attempt to guess the remote code as often as it wants. The other end will simply reply with the number of bulls and cows for each attempted guess. Each unit must count how many unique guesses it received (the same guess may come in many times if the requester did not receive the reply due to network errors).

Guess attempt format:
LEN(1);DST(1);SRC(1);'G';'<Guess Count>'(2 bytes);'<Guess value>'(4 character ASCII guess e.g.: '1234', 0-9)

Guess Reply format:
LEN(1);DST(1);SRC(1);'Y';'<B>'(1 byte, number of bulls, ASCII encoded);'<C>'(1 byte, number of cows, ASCII encoded)

If the reply indicates 4 bulls, the game is finished since the requester found all the digits in the code.

If a guest attempt is left unanswered, the unit must send the guess attempt packet repeatedly until it is acknowledged. Each of those re-tried guesses will carry the same 'Guess Count' indicator since its not the user attempting to guess again the same value, but the system trying to communicate via the noisy radio link.

Server messages

Bob and Alice want their device to be used in tournaments. The organizers want to be able to send a game termination message that will stop all games in progress and notify the users. Server messages use the packet type 'S' (the server message type in this case is 'T') on the unit screen. Other server messages (non-'T' messages) should be printed on the debug port, but not handled in any particular way. The tournament server may send many termination messages in a sequence to ensure all the stations received the message. Since the tournament server does not maintain a list of active clients (its doable, but complex) then its assumed that after sending many termination requests all the games will have been interrupted.

Server Message Format:
LEN(1);DST(1,broadcast address); SRC(1);'S';'T'(game termination);'String message' (Up to 30 characters, NULL terminated)

A programmable setting should be offered to disable the handling of termination messages. This will allow you to play one to one games while other teams may be debugging those messages on the same channel as the one you are playing on...

You could also send termination message to one receiver by using its direct address. This will allow you to stop a game in progress in a controlled manner (and test your code). However, if one unit just stops playing, the heartbeats will stop and the game will terminate the game. You could use the termination message to quit (gracefully) in the middle of a game (and send the message:'Goodbye!").

Proprietary Extensions

As you probably noticed, the packet type only use a very small subset of the 256 possible values. You are free to implement your own extensions to the packets as long as they don't change the base protocol. Before you implement an extension, you will need to post your extension identifier (the byte that describes it) on the WebCT board so that other teams don't use your team extension identifier (H,R,Z,G,Y, S are reserved for obvious reasons).

Optional Project Elements

Since many of you have different levels of interest in hardware/software elements, this lab offers possible options that are either software or hardware related. Each team **is required to select and implement one of the given options**. However, your team may want to attempt more options with the following table depicting the effects of adding options to your project

Option	Specified	Not Specified
Delivered	Bonus Marks	Positive
Not Delivered	Penalized	OK

Hardware options

- Upgrade to a graphical dot matrix LCD screen from the 2x40 characters display – You should display text and a few pixel-based graphics. A good use of graphics would be a bar graph showing the received signal strength of the last 10 packets (making use of the RSSI indicator). If this option is selected, you are required to implement it (since we have to purchase the modules).
- PC PS/2 Keyboard instead of the keypad – You would have to power the keyboard and retrieve the scan codes from the keyboard. Those scan codes have to be converted to keystrokes for your embedded system. You should support typing keys from the keypad panel of the PC keyboard. The PS/2 interface operates at 5V, so you need to pass/process those signals through the CPLD.
- Another SPI slave to the SPI bus – This hardware option requires you to add a second SPI slave to your bus in addition to supporting the CC2500. The technicians have real-time clocks, digital to analog converters and SPI memory available.
- Other ideas can be proposed as long as the hardware cost is reasonable, the parts are easy to obtain and within the capability of the MSP430 to handle.

Software options

- Use FreeRTOS to build your final project on. You should also use the RTOS primitives such as semaphores, queues, mutexes. Please limit your code to a few tasks to limit the stack usage and to avoid running out of system memory.
- Handle in-game channel switching and rate re-negotiation. You can augment the protocol to allow a dynamic frequency switch or a change in the data rate. The game master should initiate the change request. Once confirmed by the slave, the switch to the new frequency and new data rate should be attempted. A re-connection should be detected. If the new frequency + rate cannot be successfully negotiated, the system should fall back to the standard rate and channel.
- Propose your own ...

Schedule

Oct 30 – Team formation and selection of hardware and software options. Each team must elect a team leader which will be responsible for the submission of the hardware/software option document and final project submission.

Nov 12/13 – Driver demonstration. All your required drivers must me demonstrated on that date. You will have to show the TA that your team is able to operate each element of your complete solution individually. You will also have to explain how the final software will be structured and how each driver will be integrated.

Nov 25/26 – Final demonstration. In this demonstration, all the team members must be present. We will require each team to make an appointment for a 40 minute demonstration and each member must explain his/her contribution to the project and answer technical questions. A survey will be given to the team that will anonymously rank each team member's contribution. The result of this survey may result in some team members getting a grade different than their peers if the survey shows that they did not invest a comparable effort in the project.

Dec 3 – Project Report. The project report must be submitted on Dec 3 before midnight via WebCT (Electronic PDF copy) or as a paper copy (please notify the TA if you wish to submit using this method). The team leader will be responsible for submitting the final project report on behalf of the team. However, in the event that the team leader cannot submit the report on time, any member of the team can submit a copy of the report and the TAs will sort out the details. Note that the submission must be received on time to avoid penalties.

Appendix A – Protocol trace example

In this example, the source and destination are 0x0A for Alice's unit, 0x0B for Bob's unit. Alice's unit will be the game host and Bob will be the client. LEN is the byte representing the computed length of the packet (should be below or equal to 64). The letter in front of the packet indicate which unit is sending

-- Alice Turns on her unit and selects game host mode
[Note that Alice's unit uses the broadcast address]
A: LEN,0xFF,0x0A,'H','Alice\0'
A: LEN,0xFF,0x0A,'H','Alice\0'
[... and so on...]
-- Bob Turns on his unit and selects client mode. His unit will capture one of the above packet and start the negotiation process
B: LEN,0x0A,0x0B,'R','Bob\0'
A: LEN,0x0B,0x0A,'Z',0x01
A: LEN,0x0B,0x0A,'Z',0x02
B: LEN,0x0A,0x0B,'Z', 0x01
A: LEN,0x0B,0x0A,'Z', 0x01
[... they now exchange heartbeats regularly and should both be in gaming mode at this point...]
B: LEN,0x0A,0x0B,'G',0x0001,'1234'
A: LEN,0x0B,0x0A,'Y','2','2'
[... Bob just attempted guessing 1234 as the secret code, Alice's unit responded with 2 Bulls, 2 Cows]
A: LEN,0x0B,0x0A,'Z',0x02
B: LEN,0x0B,0x0A,'Z',0x01
[... Heartbeats are still exchanged regularly ...]
A: LEN,0x0B,0x0A,'G',0x0001,'0000'
B: LEN,0x0A,0x0B,'Y','0','0'
B: LEN,0x0A,0x0B,'G',0x0002,'4231'
[Alice missed that guess sent by Bob due to Wifi interference. Its OK, Bob's unit will just attempt the guess again (keeping its guess counter the same)]
B: LEN,0x0A,0x0B,'G',0x0002,'4321'
A: LEN,0x0B,0x0A,'Y','4','0'
[Bob just guessed Alice's secret code (4 Bulls, 0 Cows). Bob's unit will indicate that he won. Alice's unit will know she sent the winning results to Bob. However, she will wait for 10 seconds before exiting the game completely, to ensure that Bob's response was heard. Its possible that Bob's unit do not receive the response, thus will be sending the same guess packet until he gets the reply. Alice's unit should stop allowing Alice to send guesses.]

As you see from this simple example, there are a few holes in the protocol, especially towards the end of the game since there is no formal two-way handshake to guarantee that each unit has reached the same state. One thing is sure, is that if one unit stops sending heartbeats at any point, the other unit will keep sending its heartbeats and at some point, will realize that nobody is at the other end and will terminate its own game.

This whole protocol was contrived in order to keep things simple, yet should be reliable enough to play a game in the presence of interference. You are welcome to highlight any error conditions that you find in building your solution and share your insight with the class.