Team Cloverfield Project Report

D4 Design Project - SPECIES

Alaa Khoja (ak12g12), Diwen Hu (dh1g14), Fiona Moore (fm4g13), Joseph Sturgeon (jms2g13), Nathan Ruttley (ner1g13), Yubo Zhi (yz39g13)

1. Challenge Solution Statement

Challenges addressed

A world rife with desolation and despair presents many challenges to the surviving human population, our device addresses many of these challenges.

The first of these challenges is the need to feel and be safe, our device incorporates a hands-free system with low-latency communication through the use of a headset with integrated microphone. The concept of "safety in numbers" is well known, our device also allows groups of people to communicate with others through the use of its speakerphone facility.

Audible communication may not be appropriate in all situations, through the use of an easy-to-use touchscreen sketch pad messages can be written or drawn in the users own handwriting and transmitted to another nearby device. The benefit of using tactile input as opposed to a keyboard is that data can be input at a greater speed by the user.

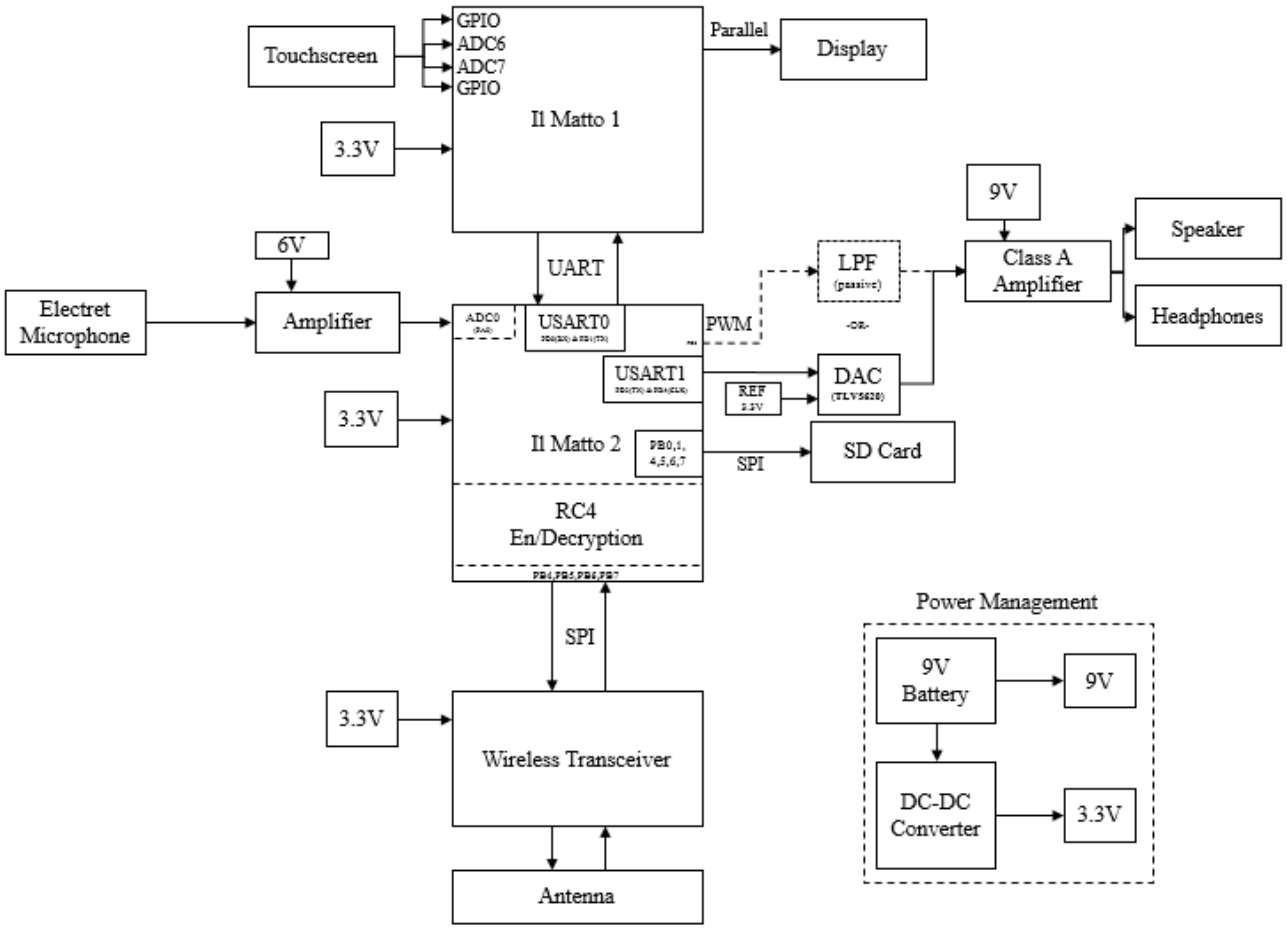
Importantly all of this communication must be wireless, our device uses a wireless module to communicate with another over a short range (tens of meters), however wireless communication is prone to interception or modification and as such our device encrypts transmitted data before it is sent.

Specification

Complete solution will be a device capable of:

- Capturing audio over a frequency range of at least that of human speech (300Hz to 3400Hz) from a microphone integrated into a headset for personal communications or a built in microphone on the device that will be used when groups of people wish to communicate with another group (speakerphone mode). The microphone amplifier will amplify voltages of approximately 20mV (peak to peak) to voltages of 3.3V (peak to peak) for use with the ADC of an Il Matto. The microphone amplifier will also have a 3dB bandwidth of at least 4kHz
- Outputting audio in a headset, again for person-to-person communication, or by loudspeaker for speakerphone mode. The audio amplifier will amplify voltages of 0 to 3.3V from the low pass filter to voltage of -8 to 8 to drive a loudspeaker, it will also have a 3dB bandwidth of at least 3100Hz, centred around 1850Hz (300 to 3400Hz for human voice, this was revised since the original specification).
- Capturing written input from a resistive touchscreen
- Displaying written data on the screen of the device
- Authenticating a user
- Encrypting and decrypting data by RC4
- Transmitting and receiving data with a packet structure over a wireless link at a length of tens of meters.

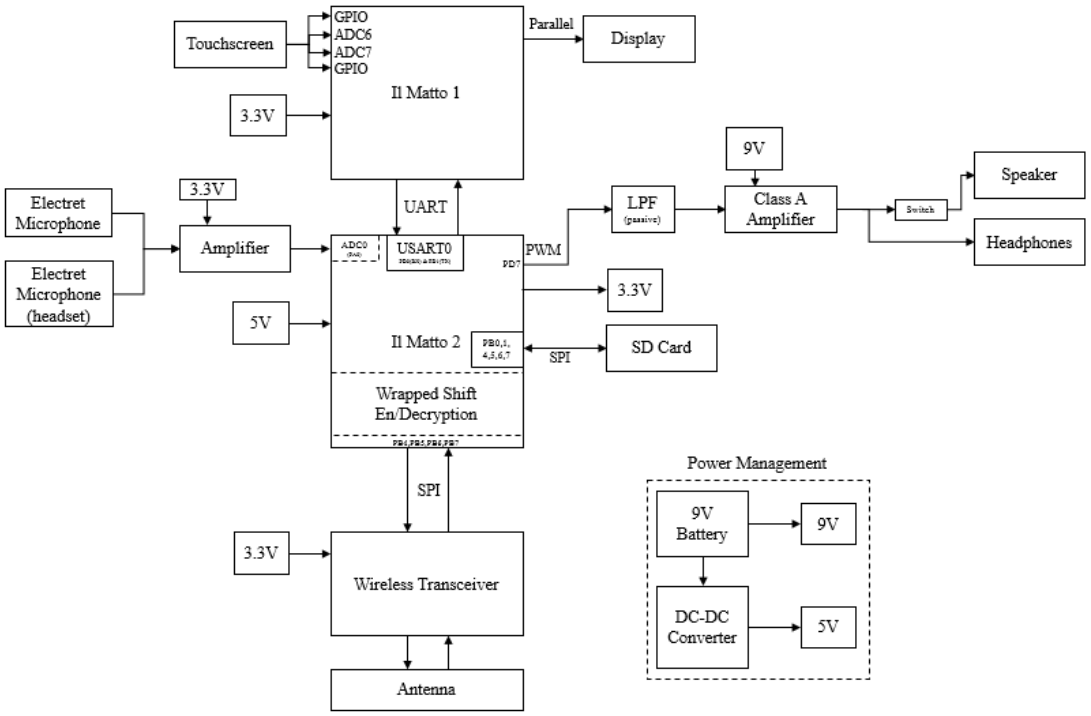
The original specification is detailed in block diagram form below



Top-level block diagram of the system specification

2. System Design (this section is based on what your team designed)

The system is comprised of two MCUs connected together. One handles data transmission, audio capture and output whilst the other captures touchscreen input and gives visual output. The audio input is from a microphone either incorporated into a headset or a microphone which can be used in conjunction with our speakerphone function. The audio outputs are complimentary to the inputs as one is through the headset with the microphone and the other is a loudspeaker to be used in the speakerphone mode. The touchscreen input/visual output MCU is wrist-mounted for ease of access whilst the other MCU is mounted on the waist. The final system is summarized in the block diagram below. Il Matto 1 is the wrist mounted module and Il Matto 2 is the waist mounted module; Il Matto 2 was originally specified to be head mounted.



Final design hierarchy.

It became evident when discussing the architecture of the device that each team member was confident programming for the AVR in C, therefore it was decided that the Il Matto would be the best choice. There was also an overall lack of familiarity with the other micro-arcana boards and given the tight deadlines required by the project it was decided that it would be a poor investment of time to learn how to use these devices properly when it was not expected that they would add any significant benefit to the end product.

3. Design Evaluation (this section is based on what your team designed)

Evaluation of the design was carried out by considering on the following points:

Difficulty of attempted specification.

The attempted specification posed challenges in several areas. The first being the use of a resistive touchscreen to capture tactile input as this was a component that had not been previously tested. The attempted specification posed challenges in several areas. The first being the use of a resistive touchscreen to capture tactile input as this was a component that had not been previously tested. The second was the use of both a headset and an external microphone and speaker, this meant that amplifiers had to be designed to ensure compatibility with multiple inputs and outputs.

To ensure that the final design met the specification set for the project it was decided to have two separate units connected together by a cable which posed further challenges in terms of data corruption over the cable and choosing master and slave devices. The transmission between the two devices also had to be encrypted which created difficulties with regards to data transfer rates. It was also decided to have the two units' wrist and waist mounted to ensure that the user's mobility was not impaired, this meant that cases had to be designed to house the devices. Overall the attempted specification was challenging in terms of both the hardware requirements, the code needed and the interfacing between the two Il Mattos, wireless modules and the other hardware.

Quality of the electronic design.

The overall quality of the build is fairly high for a prototype, however more care ought to have been taken during the final construction phase as some of the wires and the metal backing of the speaker were left uncovered. This

could cause the system to fail if any undesirable connections are made by mistake in the container and all wires and metal should be insulated (e.g. by tape) to prevent this from occurring.

Another issue with quality was one of the resistive touchscreens. Near the end of the prototyping phase of the design one of the touchscreens stopped working as effectively as it did at the start. This was believed to be due to overuse and the lack of an appropriate support structure throughout most of the design phase.

Ease of use.

The design is relatively easy to use, audio can be captured using either a mounted microphone in speakerphone mode or a headset microphone for personal usage. In addition the device provides visual and handwritten messages communicated using a resistive touchscreen that is mounted on the wrist also making the GUI easily accessible. The menu system implemented on the touchscreen device is easy to use, especially for users already familiar with using touchscreens. This familiarity is particularly important as in the war against the machines the user does not want to be learning how to use new systems or complex GUIs.

Creativity and innovation of the final product.

The design is certainly creative through the use of both tactile input and visual output in conjunction with the audio input and output, this feature was carefully considered for its benefits in the scenario with the main uses being that communication can happen quickly with very minimal sound being created or movement needing to be made. Another benefit of using this type of input is that schematics can be drawn to show escape routes and for entertainment. This ability to transmit and receive small sketches with very low latency and high accuracy is the unique selling point of the product and sets it apart from similar products.

The design is innovative through its use of very simple components and modules to create a novel yet useful experience for the user. The use of two separate modules ensures that the user is not burdened with one heavy device but rather two body-mounted and relatively light modules.

Aesthetics of the final product.

Our design is well organized, with the two modules mounted onto the wrist and waist, with only one cable between the two. The wrist mounted module is housed in a white 3D printed case which allows customization of the design by the user. The aesthetics of the wrist mounted device are more pleasing than the waist mounted one; this is due to the waist mounted device containing the majority of the peripherals, such as the speaker, which take up a considerable amount of space. No graphics were printed on the case with the exception of a transparent back panel; it was decided that functionality was far more important than form, especially given the scenario.

Cost effectiveness.

By avoiding the use of complex devices such as FPGAs and minimizing the use of surface mount components the manufacturing time of a whole device is estimated to take no longer than 6 hours. This means that not only are the components cheap but the device can be made quickly reducing production overheads. Once out of the prototyping phase the 3D printed case could be injection moulded from a tough plastic, increasing durability and reducing the device cost significantly.

Design reliability.

The device rarely experiences problems, and in the case of a software issue it can simply be power cycled to return to a working state. There is no start-up configuration needed between the two devices meaning that the only data lost is the data transmitted when the device is powered down. No complex initialisation is needed from the user apart from calibrating touchscreens however this need only be completed once and does not need to be repeated each time the device is turned on. The reliability of external circuits could be increased by ensuring no metal-to-metal contacts could be made with other circuits as short circuits can occur within the device housing.

The device rarely experiences problems, and in the case of a software issue it can simply be power cycled to return to a working state. There is no start-up configuration needed between the two devices meaning that the only data lost is the data transmitted when the device is powered down. No complex initialisation is needed from the user apart from calibrating touchscreens however this need only be completed once and does not need to be repeated each time the device is turned on.

During the prototyping phase one of the touchscreens was damaged, this causes glitches during sketching and can cause difficulty in pressing menu items.

4. Costing, Marketing and Conformance marking

Costing

The total costs for each aspect of the design are detailed in the table below

Prototyping Cost (components and materials)	122.54
Prototyping Cost (person-hours and conformance testing)	10625.00
Total Prototyping Cost	**110747.54**
Large Scale Device Cost (components and materials)	60.67
Large Scale Device Cost (construction)	60.00
Total Device Cost (large scale)	**120.67**
Sale Price (excluding VAT)	**277.77** (333.33 including VAT)
Profit (per device)	**157.105**

The profit per device and the total prototyping cost means that the breakeven sale quantity is 705, this is certainly achievable given the remaining population of Earth.

Marketing

A short video advert was created to be presented alongside a printed poster. The aim was to ensure that people from as many cultures as possible could understand the marketing as the target market does not consist only of English speakers. The focus of our marketing campaign will be the extremely low latency shared drawing function in which two users can collaborate wirelessly to draw diagrams, pictures or write handwritten messages to each other and the ability to accept or reject communications from another device..

Conformance marking

Our device falls under five CE marking directives that it needs to comply with in order to conform to legislation [1]. The five directives and the measures that need to be undertaken are as follows:

1. WEEE (Waste Electrical and Electronic Equipment) Directive:

   Producers must register with a compliance scheme to allow the waste electronic equipment to be recycled and treated though collaboration with the relevant agencies.

   This must be done alongside the marking of the product in which certain information must be provided such as the producer ID and the date when the product was produced, as well as information for treatment facilities.

2. Restriction of Hazardous Substances Directive (RoHs):

   The producer must acquire a certificate from the supplier to confirm that the components and subsystems used in the design are RoHs complaint, alongside a technical file must be produced with all of the component data and analysis, which must be retained for a period of four years from the date the device was marketed.

3. Radio & Telecommunications Terminal Equipment Directive

It must be ensured that the device does not impose interference with any orbital or terrestrial communication; the producer must either provide a declaration of conformity if certain standards are applicable for the device or alternatively an official third party must conduct an assessment of the device.

4. Electromagnetic Compatibility (EMC) Directive

   The producer has to provide a technical report to confirm that the device does not produce undesired electromagnetic radiation, along with a declaration of conformity, CE logo printed or engraved on the device, and instructions for proper usage for the user to ensure that misuse does not cause any undesirable effects.

5. General Product Safety Directive

   The producer must supply clients with necessary instruction for safe usage and notify them about possible risks that could occur when using the product.

   The producer must notify the authorities in the event of the device being found to be unsafe. The producer must have a method in place by which the product can be quickly recalled to minimise any risk or potential for harm.

5. Final Product

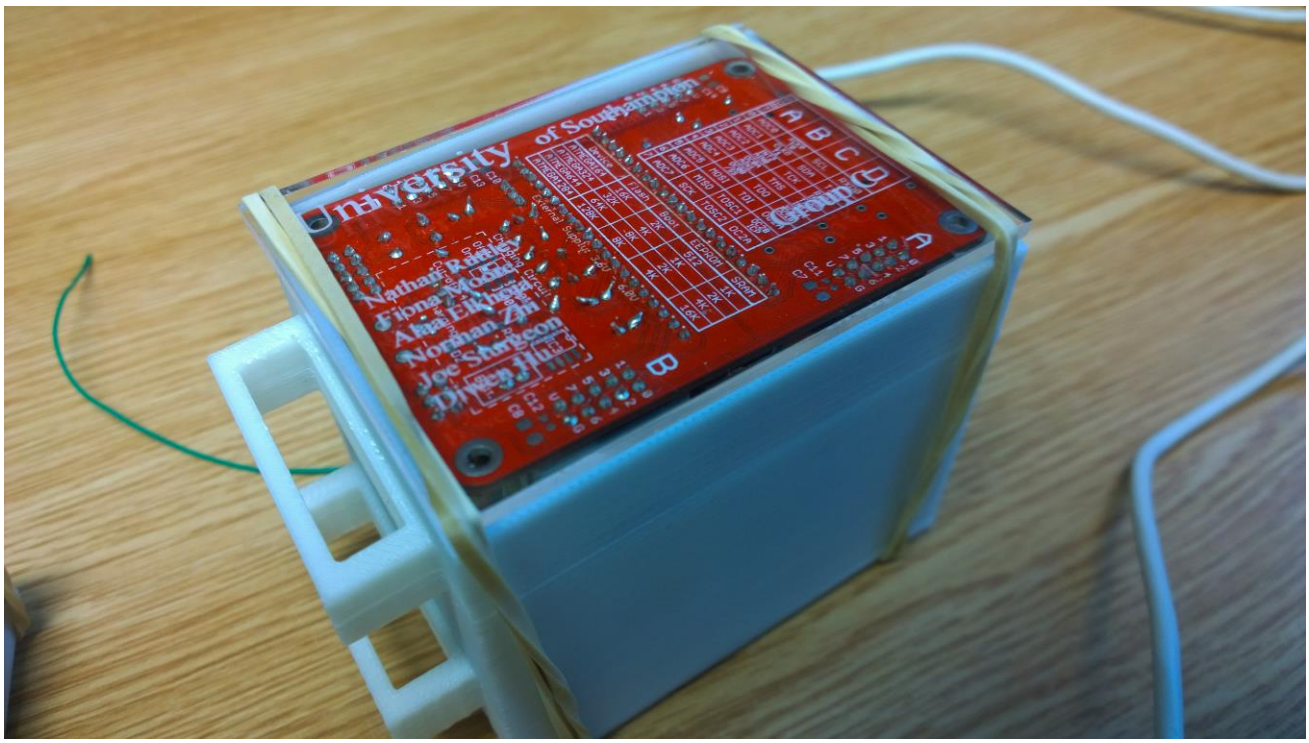
Overall the final product was very similar to the initially proposed schematic, the block diagram in particular is a fairly accurate reflection of the final product excepting the SD card. More details about the specific changes made are discussed in the Design Changes section as well as in the attached individual reports.

Regarding the wireless transceiver modules (RFM12B-S2) and their interface with the design as a whole very little was changed from the initial schematic and most of this section of the design was purely software based with the exception of a small amount of soldering and wiring. The module functions as intended, allowing both data and audio to be transmitted and received in packets with a size of 64 bytes.

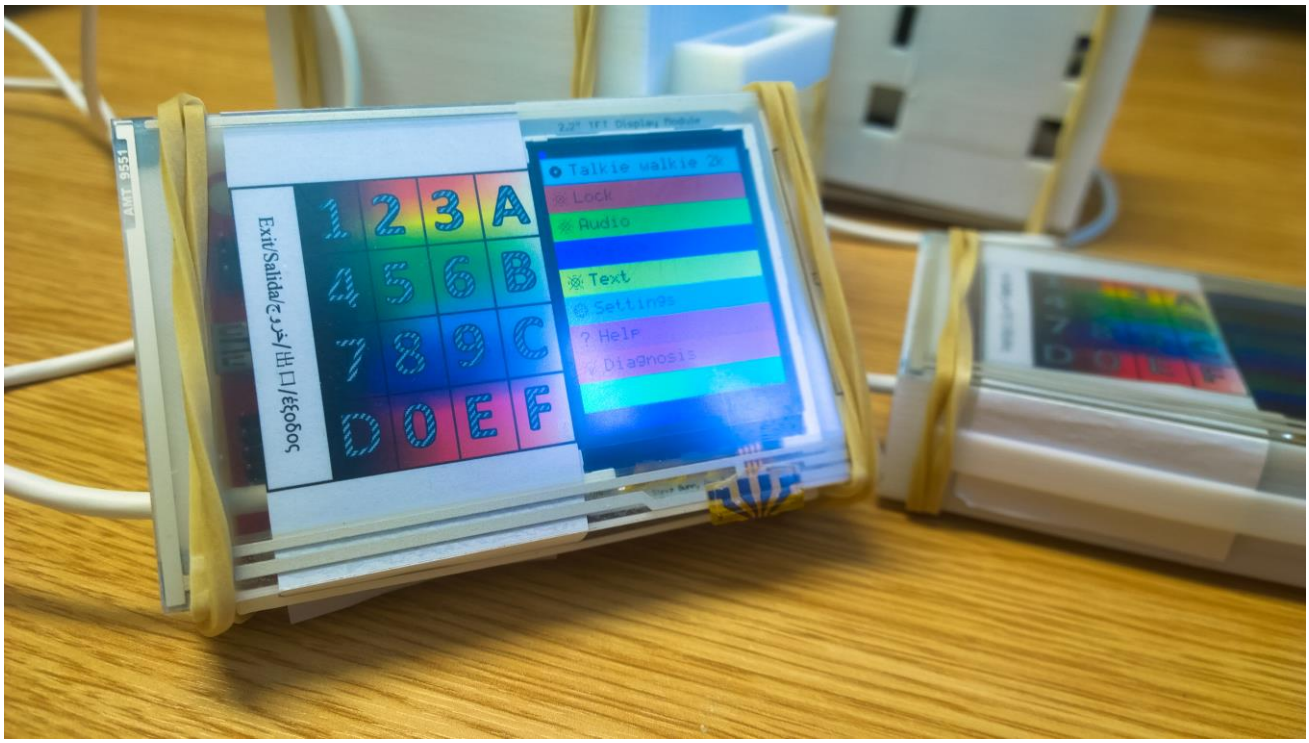
The audio output could be improved with more time, perhaps the creation of two dedicated amplifiers (one for loudspeaker audio and the other for headset audio) or one amplifier with a variable gain. Improved filtering could be introduced to remove the noise that can be heard on the output, this would likely be done through the use of active and passive filtering as none of the purely passive filters that were design performed as desired.

The microphone amplifier performs extremely well with the on-board microphone being able to pick up audio from large distances away, this feature is extremely useful in our speakerphone function. The microphone amplifier could be improved through implementing filtering to remove the noise introduced by the low quality microphone in the headset.

Images of the final product can be seen below.



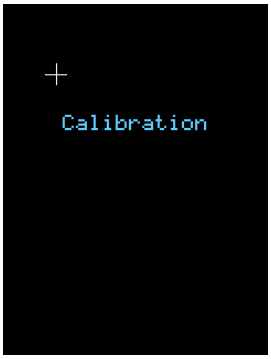
Bottom view of the waist mounted unit, the green wire on the left is the antenna.



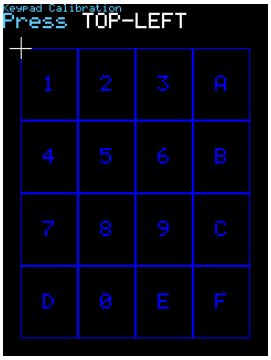
Close-up of the wrist mounted unit with the top level GUI menu displayed

Close-up of the top of the waist mounted unit showing the loudspeaker

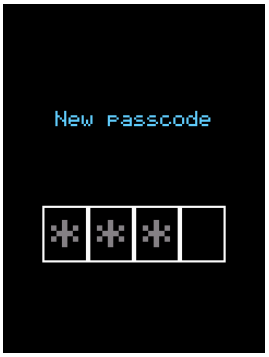
The following images are of the GUI menu and some of its functions.



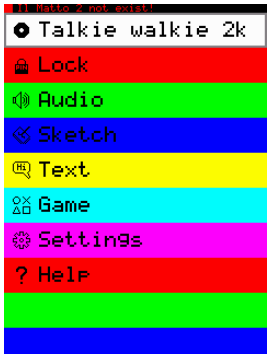
The calibration function



The virtual keyboard calibration function

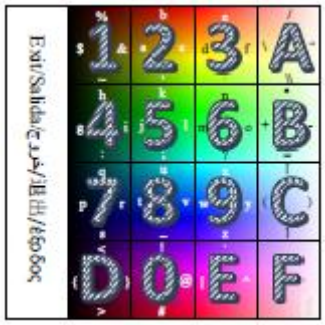


The new passcode feature



The main GUI menu

This final image is of the soft-keypad located next to the touchscreen:



The operation of the device is extremely simple; after connecting the battery the user must enter a 4 digit pin, which can be changed once logged in, to gain access to the GUI of the device; the GUI provides access to all of

the devices features including the drawing and speech functions. The audio communication is commenced through a PTT button on the GUI and users can talk aloud near the waist mounted unit (speakerphone mode) or into a headset microphone attached to the waist mounted module. The user may choose which microphone is active through the use of an easily accessible switch. Similarly, by use of another switch the audio output can be selected as either the loudspeaker or headset unit.

In addition to the main features of the device the GUI also allows users to do the following: instantly transmit touchscreen input to another device via fast half-duplex transmission (this gives the effect of full duplex), the user can also choose from 9 brush sizes and full scale colour; and write and send text messages to another device using a virtual keypad. The messages are queued at either end to ensure no loss of data which results in highly accurate communications. Given more time a help menu could be implemented to guide the user on the devices features and simple games such as tic-tac-toe could be implemented.

6. References

[1] Conformance Ltd (2014) *European Directives and Regulations*. Available from: http://www.conformance.co.uk/adirectives/doku.php [Accessed 11 March 2015]

Appendix A: Design Completion Form

Design completion form C

Component of system/Milestone	Supervisor	Time/Date	Comments (all/part/none working; protoboard/constructed)
Functioning transmit program using RFM12B modules	gvn	02/March 13:00	press button on one II Matto & thumb over
Functioning receive program using RFM12B modules	gvn	02/March 11:00	
Bi-directional short data transfers over wireless	gvn	02/March 13:00	Msg to other II Matto & [y]/n, LCD while held.
Communication between II Matto 1 and 2 using UART	QM	11:15 9th	✓ works - real time integration demonstrated.
En/Decryption of data	QM	11:15 9th	✓ works
Communication of speech between two II Mattos	QM	12:30 6th	Speech is recognisable. Amplifier built on stripboard. II Matto.
Comm of data from II Matto 1 transmitted over wireless (images)	QM	11:15 9th	✓ works - drawing on one device replicated in real time on the other.
Transmission of encrypted audio and successful decryption	QM	11:15 9th	✓ works
All wireless comms encrypted and successfully decrypted	QM	11:15 9th	✓ works
Microphone amplifier functioning	QM	5.3.15	constructed mic/amp/ADC - DAC - mic
Speech sampled by II Matto	QM	5.3.15 9th	Noise/speech limited in volume currently - good volume, but still noise
Output amplifier working with loudspeaker	QM	5.3.15	" "
II Matto input to audio amplifier can drive loudspeaker	QM	5.3.15	" "
II Matto can display on TFT	QM	9th 11:15	✓ works
Have a menu selection GUI on TFT	QM	9th 4:30	✓ works
Implemented a basic menu	QM	9th 11:15	✓ works
II Matto can capture touchscreen input from ADC	QM	9th 11:15	✓ works - uses II Mattos ADC to interpret touchscreen output
Able to translate ADC readings from touchscreen to TFT coordinate	QM	9th 11:15	✓ works
Store & load touchscreen calibration data from EEPROM	QM	9th 11:15	✓ works
Touchscreen input captured and displayed on TFT display	gvn	02/March 12:50	Draw on touchscreen & shown in real-time on LCD
Touchscreen sketching with colour selection	gvn	2/March 12:50	Can change colour of pen by clicking off screen
Touchscreen input can control menu selection GUI	gvn	2/March 12:50	Can scroll a list using touchscreen.
Combine touchscreen sketch mode from menu	QM	9th 11:15	✓ works
Control & communicate to another device through GUI	QM	9th 11:15	✓ works
Touchscreen sketch transmitted to another II Matto	QM	9th 11:15	✓ works
GUI created for user authentication	QM	9th 11:15	PIN entry required to access GUI
Able to read & write files on SD card			
Voice memos stored and retrieved on SD card			
Touchscreen sketch stored and retrieved on SD card			
Automatic microcontroller sleeping			
Power management of system	QM	9th 11:15	Two II Mattos and amp circuits running off a 9V battery.
Integration of complete system	QM	4:30 9th	Everything is in its box.
System implemented as a wearable device	QM	" "	

Appendix B: Project Completion Form

Cost Estimates

Quantity (prototyping)	Item	Price (prototype)
4	4-way RCA Jack Connector	1.54
2	Electret Microphone	1.82
2	Loudspeaker	2.3
2	Headset	4.62
2	Resistive touchscreen	25.8
2	9V Battery Connector	0.96
2	9V Battery	0.54
4	Il Matto	40.00
2	Displays	20.00
4	Dual Op amp MCP602-I/P	1.32
2	SD card reader	3.58
2	voltage regulator LM78L05ACZ/LFT1	0.94
2	DAC TLV5620IN	0.77
2	Wireless Transceiver	16.8
1	Stripboard	1.45
	Passive components	0.01
3	Transistors	0.09
Quantity (per device)	Item	Price (large scale production)
2	4-way RCA Jack Connector	0.55
1	Electret Microphone	0.56
1	Loudspeaker	0.92
1	Headset	2.31
1	Resistive touchscreen	11.87
1	9V Battery Connector	0.48
1	9V Battery	0.27
2	Il Matto	20.00
1	Displays	10.00
2	Dual Op amp MCP602-I/P	0.58
1	voltage regulator LM78L05ACZ/LFT1	0.32
1	Wireless Transceiver	11.35
1	Stripboard	1.45
	Passive components	0.01
6	Construction hours @ £10/Hour	60
Fixed Costs QTY	Item	Price
100	Software Development @ £75/hr	7500
15	Person-hours (design and debugging) @ £75/hr	1125
1	Conformance Testing	2000
1	Overheads	100000

Prototyping Cost (components and materials)	122.54
Prototyping Cost (person-hours and conformance testing)	10625.00
Total Prototyping Cost	**110747.54**
Large Scale Device Cost (components and materials)	60.67
Large Scale Device Cost (construction)	60.00
Total Device Cost (large scale)	**120.67**
Sale Price (excluding VAT)	**277.77** (333.33 including VAT)
Profit (per device)	**157.105**

The breakeven sale quantity for the device is 705 sales.

Design Changes

Many revisions were made to the original circuits to ensure they performed as intended as it was discovered in the laboratory that some of the components which were used behaved in unpredictable and unexpected ways. These revisions are particularly relevant to the amplifier circuits which received the most changes from the initially proposed specification. It was decided that ensuring the end product was of as high a quality as possible was the primary goal and therefore any modifications which improved performance and reliability should be incorporated as long as the loss of time did not detract from the ability to meet the design goals.

Through experimental testing it was decided that the audio quality was better using PWM as opposed to the DAC as was proposed in the original design. The final designs of the audio amplifiers were changed to use operational amplifiers rather than transistor amplifiers due to gain requirements.

Display orientation was changed from landscape to portrait due to the large touchscreen needing support underneath the edges to prevent touch input becoming affected by pressure from the TFT. However, this change results in a more aesthetically pleasing device and also made it easier to code the GUI.

Simple games and help contents were planned and even added to menu when designing the GUI, however they were never implemented due to time constraints. Instead, a more useful text messaging ability using a virtual keypad was implemented. This could be done relatively quickly as all the touchscreen set-up had been completed previously and the sketch data transfer protocol could be reused.

The encryption and decryption of data was originally intended to be completed using RC4. However, concerns were raised about the relative speed of this method and how it would affect the ability to transmit in real-time therefore it was decided to utilise a wrapped shifting algorithm. The code for the RC4 algorithm exists in a basic state therefore it would be possible to use it in place of the shifting algorithm and test for speed which was never actually attempted due to lack of time.

The SD card and all functions relating to it were not implemented in the final device due to time constraints. However this was always intended as an additional feature which would only be attempted when all other aspects of the device were fully functioning. Therefore everything relating to the SD card was designed in such a way that no other parts of the system would be dependent on it and no additional losses or failures would be experienced from it not being implemented.

Actual Project Activities

Activity	**Initials**	**Fri am**	**Fri pm**	**Mon am**	**Mon pm**	**Tue**	**Wed**	**Thu**	**Fri am**	**Fri pm**	**Mon am**	**Mon pm**
Building, testing and debugging microphone amplifier, and confirm the gain value.	AK	✓	✓	✓	✓							
Construction, test and adjustment of audio amplifier (on breadboard)	NER			✓	✓	✓	✓	✓				
Output audio data to DAC and test functionality - Adjust code if needed	NER					✓	✓	✓				
Modifier amplifier circuit for use with headphone jack	NER								✓			
test modified version of amplifier+ with microphone	AK			✓		✓	✓					
Build finalised design of amplifier and input data to ADC through microphone.	AK			✓	✓	✓	✓	✓	✓			
Write and test the receive function for wireless module	FM	✓	✓									
Write and test the transmit function for wireless module	JMS	✓	✓									
Integrate test and receive functions	FM			✓								
Integrate test and receive functions	JMS			✓								
Test transmitting and receiving a sine wave	FM				✓	✓	✓	✓	✓	✓		
En/decryption	JMS										✓	✓
Communication between both Il Mattos (UART)	YZ	✓				✓	✓	✓	✓			
Build RC LPF and CR HPF	DH			✓	✓		✓					
Build and test DC-DC converter and back up in/out put amplifier circuit.	DH	✓		✓	✓		✓	✓	✓			
3D printer design and laser cutting	DH								✓	✓	✓	✓
Elementary touch screen interface design	DH	✓	✓	✓	✓							
Touch screen interface and sketch design	YZ			✓	✓		✓	✓	✓	✓	✓	✓
GUI design	YZ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Test transmitting and receiving touchscreen sketch	YZ								✓		✓	
Test DAC chip output	YZ		✓			✓						
Soldering of final circuits	NER								✓	✓	✓	✓
Oversee integration of entire system	NER			✓		✓		✓	✓	✓	✓	✓

Discrepancy in Project Activities

The GUI design throughout the entire project was different to the original plan, new ideas for the GUI and its functionality such as menu icons, sketch mode full scale colour selection, and virtual keypad input meant that more time was spent developing the GUI than planned. However these features added good functionality to the final product so it was a good investment of time.

Soldering of the final circuits took place much earlier than expected as to design the final case it was necessary to know the size of the final circuits.

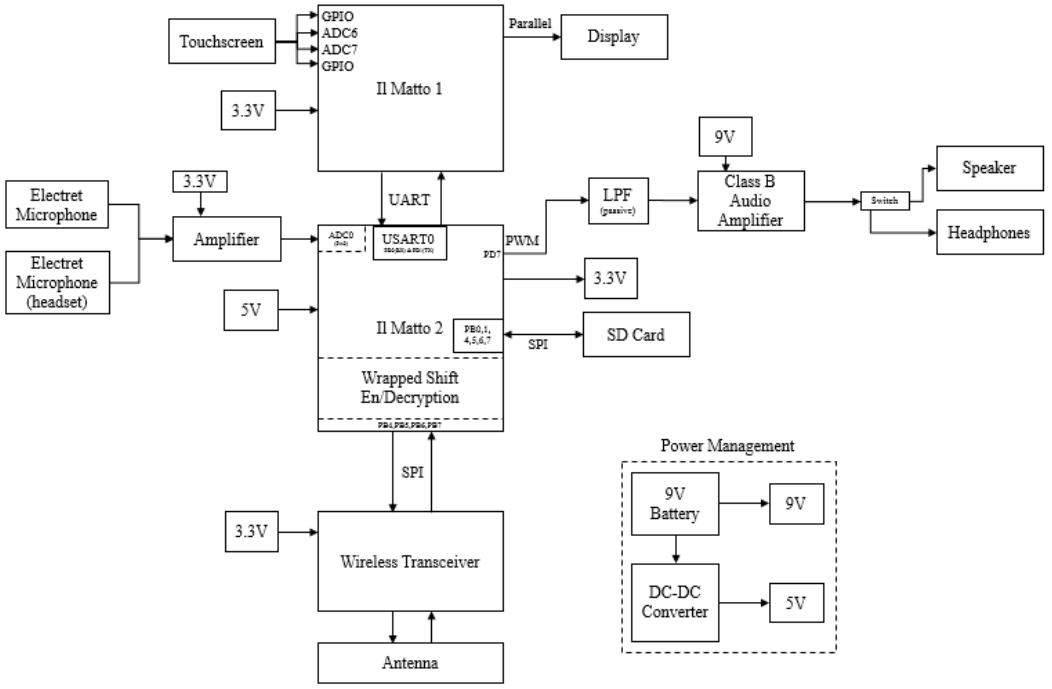
Assessment of Effort

The table below will be used as an indication how team marks should be allocated across the team.

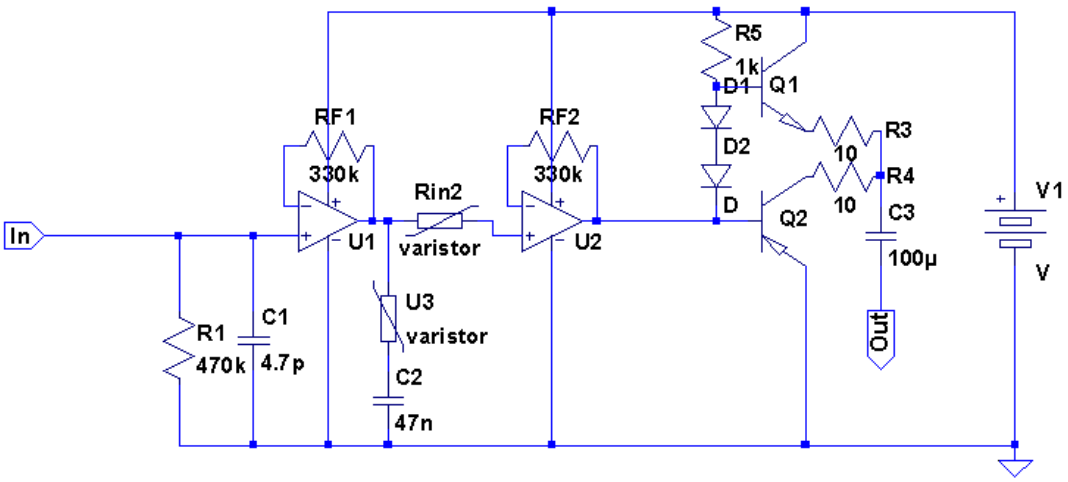
Name	**Signature**	**% of effort**
Alaa Khoja		1/6
Diwen Hu		1/6
Fiona Moore		1/6
Joe Sturgeon		1/6
Nathan Ruttley		1/6
Yubo Zhi		1/6

Appendix C: Circuit Diagrams

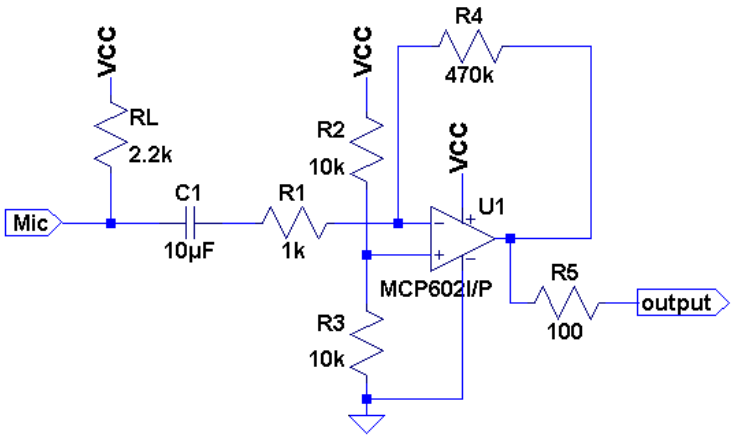
The circuit diagrams in this appendix are labelled with their corresponding name in the *Final design hierarchy* as shown below.



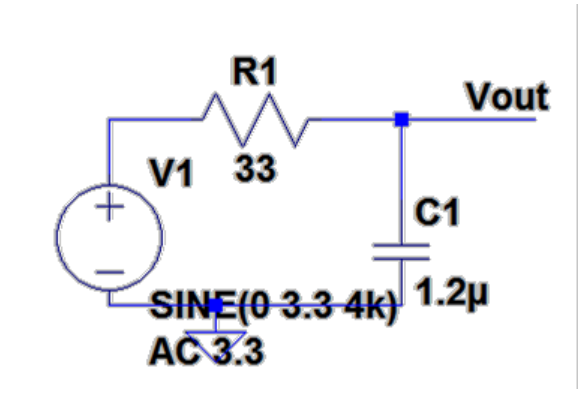
Final design hierarchy.



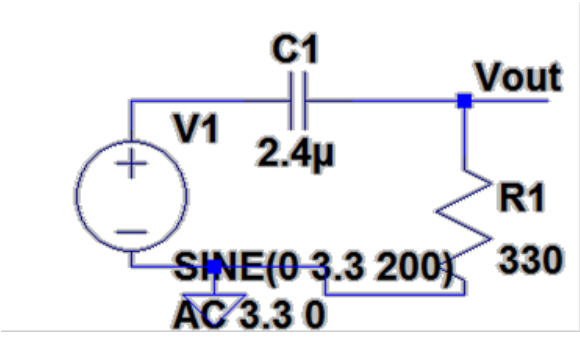
Class B Audio Amplifier



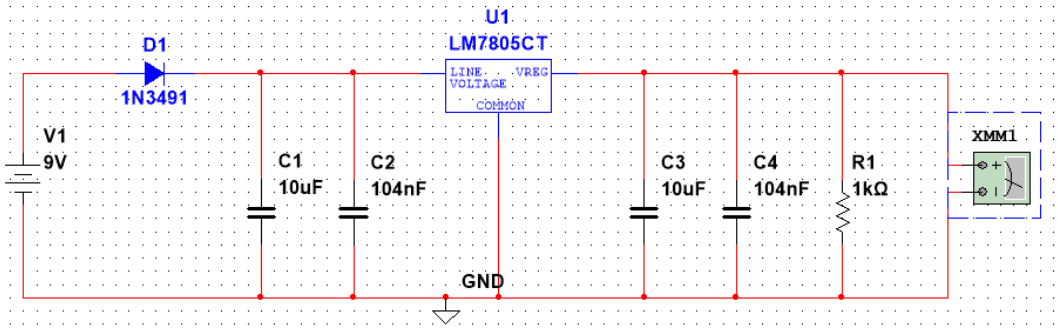
Microphone Operational Amplifier (labelled "Amplifier" in design hierarchy)



4k Low pass filter



200 High pass filter (used in conjunction with the LPF to remove high frequency noise)



9V to 5v DC to DC converter

Appendix D: Project Meeting Agendas & Minutes

Date	Topic	Actions	Notes
19/2/2015	Initial ideas discussion. Assessment of individual strengths. Nomination of team leader.	NR to create Facebook group. Each team member to read project handbook and come up with initial ideas by 20/02/2015.	All team members engaged with the Facebook page and some initial ideas were suggested in regards to the touchscreen and feasibility of this.
20/2/2015	Creation of top-level design Finalisation of features to implement. Division of work between team members	NR to digitise TLD drawing. FM and JS to conduct initial investigations in to RFM module. YZ to conduct initial investigations in to resistive touchscreen. DH to investigate power management and LPF. AK to investigate electret microphone amplification. NR to investigate loudspeaker amplification.	Discussion was had between team members in order to ascertain roughly which components were required.
23/2/2015	Pre-design clinic meeting Post design clinic meeting	Clarification of top-level design. All team members to submit required components by 24/2/2015 PM.	 Component list collated by the evening of 22/2/2015
24/2/2015	Finalise Project Scope and expected outcomes	All team members to confirm what functions their subsystems will carry out.	
26/2/2015	Finalising Project Proposal Form	Each team member to fill out in detail the specification for their subsystem.	This must be completed by 10pm 26/2/2015.
28/2/2015	Discuss changes to project completion form	Each team member to edit their section of PCF	
4/3/2015	Discuss current progress and integration	All code to be written with interrupts (Il Matto 2) for easy integration.	
7/3/2015	Meet to discuss integration and final design		YZ, FM and JS to complete integration of code on 8/3/2015
8/3/2015	Meet to decide on team report responsibilities		All team members aware that they need to contribute to team report

Appendix D: Software Listings

Library written by yz39g13 and used in this project can be found at https://github.com/zhiyb/Il-Matto.

The Library used for the RFM modules can be found at https://www.das-labor.org/storage/LaborLib/rfm12/tags/rfm12-1.1/

Listing 1: IlMatto1/common.h

```
/*
 *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#ifndef COMMON_H
#define COMMON_H

#include <tft.h>
#include <colours.h>
#include <rtouch.h>
#include <portraitlist.h>
#include <communication.h>
#include "sketch.h"
#include "keypad.h"
#include "status.h"
#include "indicator.h"
#include "notification.h"
#include "pin.h"
#include "uart0.h"
#include "tick.h"
#include "package.h"
#include "pool.h"

#define TICK_CLEAR           0
#define TICK_PING            10
#define TICK_PING_CHECK      15
#define TICK_PING_REMOTE     10
#define TICK_PING_REMOTE_CHECK (TICK_PING_REMOTE + COM_W_PING_TIMEOUT + 10)

using namespace colours::b16;

extern tft_t tft;
extern rTouch touch;
extern PortraitList list;
extern sketch_t sketch;
extern keypad_t keypad;
extern status_t status;
extern pin_t pin;
extern notification_t notification;

#endif
```

Listing 2: IlMatto1/element/indicator.cpp

```
/*
 *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#include "indicator.h"
#include "common.h"

#define STATUS_X      (0)
#define STATUS_TEXT_W (tft.width() - INDICATOR_SIZE - STATUS_X)
#define COLOURPICKER_X (INDICATOR_SIZE)

void indicator::refresh(const uint16_t clr, const char *str)
{
        tft.rectangle(STATUS_X, 0, INDICATOR_SIZE, INDICATOR_SIZE, clr);
        if (!str)
                return;
        tft.rectangle(STATUS_X + INDICATOR_SIZE, 0, STATUS_TEXT_W, INDICATOR_SIZE, Black);
        tft.setZoom(1);
        tft.setForeground(clr);
```

```
20          tft.setBackground(Black);
21          tft.setXY(INDICATOR_SIZE + INDICATOR_SIZE / 2, 0);
22          tft.putString(str, true);
23  }
24
25  void indicator::colourPicker(const uint16_t clr, uint8_t size, const bool refresh)
26  {
27          if (size > INDICATOR_SIZE)
28                  size = INDICATOR_SIZE;
29          if (refresh)
30                  tft.rectangle(COLOURPICKER_X, 0, INDICATOR_SIZE, INDICATOR_SIZE, Black);
31          tft.rectangle(COLOURPICKER_X + (INDICATOR_SIZE - size) / 2, (INDICATOR_SIZE - size) / 2, size,
                size, clr);
32  }
33
34  void indicator::checkIlMatto2(bool detailed)
35  {
36          if (tick() < TICK_PING_CHECK)
37                  return;
38          if (!status.exist.IlMatto2Updated)
39                  return;
40          if (status.exist.IlMatto2)
41                  indicator::refresh(Blue, detailed ? PSTR("") : 0);
42          else
43                  indicator::refresh(Red, detailed ? PSTR("Il Matto 2 not exist!") : 0);
44  }
45
46  void indicator::checkRemote(bool detailed)
47  {
48          if (tick() < TICK_PING_REMOTE_CHECK)
49                  return;
50          if (!status.exist.remoteUpdated)
51                  return;
52          if (status.exist.IlMatto2) {
53                  if (status.exist.remote)
54                          indicator::refresh(Green, detailed ? PSTR("") : 0);
55                  else
56                          indicator::refresh(Blue, detailed ? PSTR("Remote not exist!") : 0);
57          } else
58                  indicator::refresh(Red, detailed ? PSTR("Il Matto 2 not exist!") : 0);
59  }
```

Listing 3: IlMatto1/element/indicator.h

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #ifndef INDICATOR_H
6   #define INDICATOR_H
7
8   #define INDICATOR_SIZE LIST_TOP_RESERVED
9
10  #include <inttypes.h>
11
12  namespace indicator
13  {
14          // Existence checking
15          void checkIlMatto2(bool detailed);
16          void checkRemote(bool detailed);
17
18          // String in PROGMEM
19          void refresh(const uint16_t clr, const char *str);
20
21          // Colour picker display
```

```
22          void colourPicker(const uint16_t clr, uint8_t size, const bool refresh = true);
23  }
24
25  #endif
```

Listing 4: IlMatto1/element/keypad.cpp

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #include <eemem.h>
6   #include <rgbconv.h>
7   #include "keypad.h"
8   #include "common.h"
9
10  // Calibration cross size
11  #define CALIB_SIZE    10
12
13  // Keypad drawing position
14  #define DRAW_X        16
15  #define DRAW_Y        40
16  #define DRAW_W        (tft.width() - DRAW_X * 2)
17  #define DRAW_H        (tft.height() - DRAW_Y - 16)
18  #define DRAW_KEY_W    (DRAW_W / KEYPAD_SIZE_X)
19  #define DRAW_KEY_H    (DRAW_H / KEYPAD_SIZE_Y)
20  #define DRAW_NAME_X   ((DRAW_KEY_W - FONT_WIDTH * 2) / 2)
21  #define DRAW_NAME_Y   ((DRAW_KEY_H - FONT_HEIGHT * 2) / 2)
22
23  const char keypad_t::PGMkeyName[KEYPAD_SIZE] PROGMEM = {
24          '1', '2', '3', 'A',
25          '4', '5', '6', 'B',
26          '7', '8', '9', 'C',
27          'D', '0', 'E', 'F',
28  };
29
30  const uint8_t keypad_t::PGMkeyCode[KEYPAD_SIZE] PROGMEM = {
31          0x01, 0x02, 0x03, 0x0A,
32          0x04, 0x05, 0x06, 0x0B,
33          0x07, 0x08, 0x09, 0x0C,
34          0x0D, 0x00, 0x0E, 0x0F,
35  };
36
37  const char keypad_t::PGMkeyText[KEYPAD_SIZE - 1][5] PROGMEM = {
38          {'0', ' ', '!', '@', '#',},  // 0
39          {'1', '$', '%', '&', '~',},  // 1
40          {'2', 'a', 'b', 'c', ',',},  // 2
41          {'3', 'd', 'e', 'f', '.',},  // 3
42          {'4', 'g', 'h', 'i', ':',},  // 4
43          {'5', 'j', 'k', 'l', ';',},  // 5
44          {'6', 'm', 'n', 'o', '?',},  // 6
45          {'7', 'p', 'q', 'r', 's',},  // 7
46          {'8', 't', 'u', 'v', '_',},  // 8
47          {'9', 'w', 'x', 'y', 'z',},  // 9
48          {(char)(uint8_t)128, '\'', '/', '"', '\\',}, // A
49          {(char)(uint8_t)129, '+', '*', '-', '=',}, // B
50          {(char)(uint8_t)130, '(', '[', ')', ']',}, // C
51          {' ', '{', '<', '}', '>',},  // D
52          {' ', '|', '`', '^', ' ',},  // E
53  };
54
55  struct keypad_t::cal_t EEMEM keypad_t::NVcal;
56
57  void keypad_t::init(void)
58  {
```

```
59          prevTest = prev = KEYPAD_NA;
60  }
61
62  uint8_t keypad_t::translate(uint8_t idx)
63  {
64          return pgm_read_byte(&PGMkeyCode[idx]);
65  }
66
67  void keypad_t::drawCross(const rTouch::coord_t pos, uint16_t c) const
68  {
69          tft.rectangle(pos.x - CALIB_SIZE, pos.y, CALIB_SIZE * 2, 1, c);
70          tft.rectangle(pos.x, pos.y - CALIB_SIZE, 1, CALIB_SIZE * 2, c);
71  }
72
73  void keypad_t::display(void) const
74  {
75          tft.setZoom(2);
76          tft.clean();
77          tft.setOrient(tft_t::Portrait);
78          drawKeypad();
79  }
80
81  void keypad_t::calibrate(bool reset)
82  {
83          if (!reset && !eeprom_first()) {
84                  eeprom_read_block(&cal, &NVcal, sizeof(NVcal));
85                  return;
86          }
87
88          display();
89          tft.setBackground(0x0000);
90          tft.setForeground(0x667F);
91          tft.setXY(0, 0);
92          tft.setZoom(1);
93          tft.putString(PSTR("Keypad Calibration\n"), true);
94          tft.setZoom(2);
95
96          tft.putString(PSTR("Press "), true);
97          tft.setForeground(White);
98          tft.putString(PSTR("TOP-LEFT"), true);
99
100         drawCross(coordinate(0), White);
101         cal.pos = touch.waitForPress();
102         drawCross(coordinate(0), Black);
103         drawKey(0);
104
105         tft.setXY(0, FONT_HEIGHT);
106         tft.putString(PSTR("Press "), true);
107         tft.setForeground(White);
108         tft.putString(PSTR("BOTTOM-RIGHT"), true);
109
110         rTouch::coord_t pos = coordinate(KEYPAD_SIZE + KEYPAD_SIZE_X - 1);
111         pos.x += DRAW_KEY_W;
112         drawCross(pos, White);
113         cal.size = touch.waitForPress();
114         cal.size.x -= cal.pos.x;
115         cal.size.y -= cal.pos.y;
116
117         eeprom_update_block(&cal, &NVcal, sizeof(NVcal));
118 }
119
120 rTouch::coord_t keypad_t::coordinate(uint8_t idx) const
121 {
```

```
122         rTouch::coord_t pos;
123         pos.x = (idx % KEYPAD_SIZE_X) * DRAW_KEY_W + DRAW_X;
124         pos.y = (idx / KEYPAD_SIZE_X) * DRAW_KEY_H + DRAW_Y;
125         return pos;
126 }
127
128 void keypad_t::drawKey(uint8_t idx, uint16_t clr) const
129 {
130         rTouch::coord_t pos = coordinate(idx);
131         tft.setForeground(clr);
132         tft.frame(pos.x, pos.y, DRAW_KEY_W, DRAW_KEY_H, 1, clr);
133         tft.setXY(pos.x + DRAW_NAME_X, pos.y + DRAW_NAME_Y);
134         tft << (char)pgm_read_byte(&PGMkeyName[idx]);
135 }
136
137 void keypad_t::drawKeypad(void) const
138 {
139         for (uint8_t i = 0; i < KEYPAD_SIZE; i++)
140                 drawKey(i);
141 }
142
143 uint8_t keypad_t::keyAt(const rTouch::coord_t pos) const
144 {
145         if (outside(pos))
146                 return KEYPAD_NA;
147         return (pos.y - cal.pos.y) * KEYPAD_SIZE_Y / cal.size.y * KEYPAD_SIZE_X + (pos.x - cal.pos.x)
              * KEYPAD_SIZE_X / cal.size.x;
148 }
149
150 uint8_t keypad_t::pool(bool keep, bool code)
151 {
152         uint8_t idx;
153         if (touch.pressed()) {
154                 idx = keyAt(touch.position());
155                 if (keep || idx != prev)
156                         goto ret;
157                 return KEYPAD_NA;
158         } else
159                 idx = KEYPAD_NA;
160 ret:
161         prev = idx;
162         return code ? (idx == KEYPAD_NA ? KEYPAD_NA : translate(idx)) : idx;
163 }
164
165 bool keypad_t::testPool(void)
166 {
167         if (touch.pressed())
168                 if (touch.position().x >= 0) {
169                         prevTest = prev = KEYPAD_NA;
170                         return false;
171                 }
172         tft.setZoom(2);
173         uint8_t idx = pool(true, false);
174         if (idx != prevTest) {
175                 if (prevTest != KEYPAD_NA)
176                         drawKey(prevTest);
177                 tft.setXY(DRAW_X, DRAW_Y - FONT_HEIGHT * 2);
178                 if (idx != KEYPAD_NA) {
179                         uint8_t code = translate(idx);
180                         tft << (code < 0x0A ? code + '0' : code - 0x0A + 'A');
181                         drawKey(idx, KEYPAD_ACT_CLR);
182                 } else
183                         tft << " ";
```

```
184         }
185         prevTest = idx;
186         return true;
187 }
188
189 bool keypad_t::outside(const rTouch::coord_t pos) const
190 {
191         return pos.x < cal.pos.x || pos.y < cal.pos.y || \
192                 pos.x >= cal.pos.x + cal.size.x || pos.y >= cal.pos.y + cal.size.y;
193 }
194
195 bool keypad_t::colourPicker(rTouch::coord_t pos, uint16_t &clr) const
196 {
197         if (outside(pos))
198                 return false;
199         uint8_t red = 0, green = 0, blue = 0;
200         uint16_t y = (uint32_t)(pos.y - cal.pos.y) * 6 * 256 / (uint32_t)cal.size.y;
201         uint8_t c = y % 256, reg = y / 256;
202         switch (reg) {
203         case 0:
204                 red = 0xFF;
205                 green = c;
206                 break;
207         case 1:
208                 red = 0xFF - c;
209                 green = 0xFF;
210                 break;
211         case 2:
212                 green = 0xFF;
213                 blue = c;
214                 break;
215         case 3:
216                 green = 0xFF - c;
217                 blue = 0xFF;
218                 break;
219         case 4:
220                 red = c;
221                 blue = 0xFF;
222                 break;
223         case 5:
224                 red = 0xFF;
225                 blue = 0xFF - c;
226                 break;
227         }
228         uint16_t x = (uint32_t)(pos.x - cal.pos.x) * 2 * 256 / (uint32_t)cal.size.x;
229         if (x / 256 == 0) {
230                 red = (uint16_t)red * x / 256;
231                 green = (uint16_t)green * x / 256;
232                 blue = (uint16_t)blue * x / 256;
233         } else {
234                 x %= 256;
235                 red += (uint16_t)(0xFF - red) * x / 256;
236                 green += (uint16_t)(0xFF - green) * x / 256;
237                 blue += (uint16_t)(0xFF - blue) * x / 256;
238         }
239         clr = conv::c32to16(conv::c32(red, green, blue));
240         return true;
241 }
242
243 char keypad_t::text(void)
244 {
245         rTouch::Status s;
246         if ((s = touch.status()) != rTouch::Idle) {
```

```
247                 if (prev && index == KEYPAD_NA)
248                         return -1;
249                 rTouch::coord_t pos = touch.position();
250                 status = s;
251                 if (prev == 0) {
252                         first = pos;
253                         index = translate(keyAt(pos));
254                         prev = 1;
255                 }
256                 prevCoord = pos;
257         } else if (prev) {
258                 prev = 0;
259                 if (index == KEYPAD_NA)
260                         return -1;
261                 uint8_t dir = 0;
262                 if (status == rTouch::Moved) {
263                         if (abs(prevCoord.x - first.x) > abs(prevCoord.y - first.y)) {
264                                 if (prevCoord.x < first.x)
265                                         dir = 1;
266                                 else
267                                         dir = 3;
268                         } else {
269                                 if (prevCoord.y < first.y)
270                                         dir = 2;
271                                 else
272                                         dir = 4;
273                         }
274                 }
275                 if (index == 0x0F)
276                         return KEYPAD_DEL;
277                 return pgm_read_byte(&PGMkeyText[index][dir]);
278         }
279         return -1;
280 }
```

Listing 5: IlMatto1/element/keypad.h

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #ifndef KEYPAD_H
6  #define KEYPAD_H
7
8  // Horizontal size
9  #define KEYPAD_SIZE_X 4
10 // Vertical size
11 #define KEYPAD_SIZE_Y 4
12 // Index size
13 #define KEYPAD_SIZE   (KEYPAD_SIZE_X * KEYPAD_SIZE_Y)
14
15 // Keypad default display colour
16 #define KEYPAD_DEF_CLR colours::b16::Blue
17 // Keypad activative display colour
18 #define KEYPAD_ACT_CLR colours::b16::Red
19
20 // Key code for N/A
21 #define KEYPAD_NA     0xFF
22 // Key code for delete
23 #define KEYPAD_DEL    127
24
25 #include <avr/pgmspace.h>
26 #include <rtouch.h>
27 #include <colours.h>
28
```

```
29 class keypad_t
30 {
31 public:
32         void init(void);
33         void calibrate(bool reset = false);
34         void recalibrate(void) {calibrate(true);}
35         void display(void) const;
36         // Arg: keep return pressing key, Ret: index of pressed key or KEYPAD_NA
37         uint8_t pool(bool keep = false, bool code = true);
38         bool testPool(void);
39         bool colourPicker(rTouch::coord_t pos, uint16_t &clr) const;
40         bool outside(const rTouch::coord_t pos) const;
41         bool outsideLeft(const int16_t x) const {return x < cal.pos.x;}
42         uint8_t keyAt(const rTouch::coord_t pos) const;
43         // Translate key index to key code
44         uint8_t translate(uint8_t idx);
45         void initText(void) {prev = 0;}
46         char text(void);
47
48 private:
49         // Convert index to coordinate relative to TOP-LEFT for DISPLAY
50         rTouch::coord_t coordinate(uint8_t idx) const;
51         void drawCross(const rTouch::coord_t pos, uint16_t c) const;
52         void drawKey(uint8_t idx, uint16_t clr = KEYPAD_DEF_CLR) const;
53         void drawKeypad(void) const;
54
55         uint8_t prev, prevTest, index, status;
56         rTouch::coord_t first, prevCoord;
57         struct cal_t {
58                 rTouch::coord_t pos, size;
59         } cal;
60         static struct cal_t EEMEM NVcal;
61
62         // Display name (1 char)
63         static const char PGMkeyName[KEYPAD_SIZE] PROGMEM;
64         // Key code
65         static const uint8_t PGMkeyCode[KEYPAD_SIZE] PROGMEM;
66         // Text
67         static const char PGMkeyText[KEYPAD_SIZE - 1][5] PROGMEM;
68 };
69
70 #endif
```

Listing 6: IlMatto1/element/notification.cpp

```
1  /*
2   * Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #include <string.h>
6  #include "notification.h"
7  #include "common.h"
8
9  #define TEXT_ZOOM     2
10 #define TEXT_X        8
11 #define TEXT_Y        (tft.height() / 4)
12
13 #define BUTTON_X      0
14 #define BUTTON_Y      (tft.height() * 3 / 4)
15 #define BUTTON_W      (tft.width() / 2)
16 #define BUTTON_H      (tft.height() - BUTTON_Y)
17 #define BUTTON_A_CLR  Green
18 #define BUTTON_R_CLR  Red
19
20 #define BUTTON_T_X(l) ((BUTTON_W - FONT_WIDTH * TEXT_ZOOM * l) / 2)
```

```
#define BUTTON_T_Y    ((BUTTON_H - FONT_HEIGHT * TEXT_ZOOM) / 2)

#define BUTTON_TEXT_X 0
#define BUTTON_TEXT_Y (tft.height() * 3 / 4)
#define BUTTON_TEXT_W (tft.width())
#define BUTTON_TEXT_H (tft.height() - BUTTON_TEXT_Y)
#define BUTTON_TEXT_CLR Green

#define BUTTON_TEXT_T_X(l)   ((BUTTON_TEXT_W - FONT_WIDTH * TEXT_ZOOM * l) / 2)
#define BUTTON_TEXT_T_Y      ((BUTTON_TEXT_H - FONT_HEIGHT * TEXT_ZOOM) / 2)

namespace name
{
        const char PROGMEM sketch[] = "Shared sketch";
        const char PROGMEM audio[] = "Audio";
}

const char *requestName[PKG_REQUEST_COUNT] = {name::sketch, name::audio};

void notification_t::init(void)
{
        reqCnt = 0;
        reqs = 0;
        msgCnt = 0;
        msgs = 0;
        msgIdx = 0;
}

void notification_t::pushRequest(pkgRequest_t *req)
{
        request_t **ptr = &reqs;
        while (*ptr) {
                if ((*ptr)->req == req->s.req)
                        return;
                ptr = &(*ptr)->next;
        }
        request_t *p = new request_t;
        p->req = req->s.req;
        p->next = 0;
        *ptr = p;
        reqCnt++;
}

notification_t::request_t *notification_t::popRequest(void)
{
        if (!reqs)
                return 0;
        request_t *req = reqs;
        reqs = reqs->next;
        reqCnt--;
        return req;
}

void notification_t::pushMessage(pkgMessage_t *msg)
{
        message_t **ptr = &msgs;
        while (*ptr) {
                if ((*ptr)->idx == msg->s.idx)
                        return;
                ptr = &(*ptr)->next;
        }
        message_t *p = new message_t;
        p->idx = msg->s.idx;
```

```
        for (uint8_t i = 0; i < PKG_TEXT_LENGTH; i++)
                p->str[i] = msg->s.str[i];
        p->next = 0;
        *ptr = p;
        msgCnt++;
}

notification_t::message_t *notification_t::popMessage(void)
{
        if (!msgs)
                return 0;
        message_t *msg = msgs;
        msgs = msgs->next;
        msgCnt--;
        return msg;
}

void notification_t::removeRequests(uint8_t req)
{
        request_t **ptr = &reqs;
        while (*ptr) {
                if ((*ptr)->req == req) {
                        request_t *p = *ptr;
                        *ptr = p->next;
                        delete p;
                        reqCnt--;
                } else
                        ptr = &(*ptr)->next;
        }
}

package_t *notification_t::pool(package_t *pkg)
{
        reqAck = PKG_REQUEST_INVALID;
        msgAck = false;
        if (!pkg || (pkg->command != COM_W_RECV && pkg->command != COM_W_SEND))
                return pkg;
        switch (pkg->data[0]) {
        case PKG_TYPE_REQUEST: {
                        pkgRequest_t *req = (pkgRequest_t *)pkg->data;
                        uint8_t r = req->s.req;
                        pushRequest(req);
                        uart0_done(pkg);
                        sendRequestAck(r, PKG_REQUEST_RECEIVED);
                        return 0;
                }
        case PKG_TYPE_REQUEST_ACK: {
                        pkgRequestAck_t *ack = (pkgRequestAck_t *)pkg->data;
                        if (ack->s.ack == PKG_REQUEST_CLOSED)
                                removeRequests(ack->s.req);
                        reqType = ack->s.req;
                        reqAck = ack->s.ack;
                        uart0_done(pkg);
                        return 0;
                }
        case PKG_TYPE_TEXT: {
                        pkgMessage_t *msg = (pkgMessage_t *)pkg->data;
                        uint8_t i = msg->s.idx;
                        pushMessage(msg);
                        uart0_done(pkg);
                        sendMessageAck(i);
                        return 0;
                }
```

```
147         case PKG_TYPE_TEXT_ACK: {
148                         pkgMessageAck_t *ack = (pkgMessageAck_t *)pkg->data;
149                         msgAckIdx = ack->s.idx;
150                         msgAck = true;
151                         uart0_done(pkg);
152                         return 0;
153                 }
154         }
155         return pkg;
156 }
157
158 bool notification_t::show(void)
159 {
160         message_t *msg = popMessage();
161         if (!msg)
162                 goto request;
163         pool::message(msg->str);
164         delete msg;
165         return true;
166 request:
167         request_t *req = popRequest();
168         if (!req)
169                 return false;
170         pool::request(req->req);
171         delete req;
172         status.request.refresh = true;
173         return true;
174 }
175
176 void notification_t::sendRequest(uint8_t req)
177 {
178         package_t *pkg;
179         while (!(pkg = uart0_txPackage()));
180         pkg->command = COM_W_SEND;
181         pkg->length = 2;
182         pkgRequest_t *request = (pkgRequest_t *)pkg->data;
183         request->s.type = PKG_TYPE_REQUEST;
184         request->s.req = req;
185         pkg->valid++;
186         uart0_send();
187 }
188
189 void notification_t::sendRequestAck(uint8_t req, uint8_t ack)
190 {
191         package_t *pkg;
192         while (!(pkg = uart0_txPackage()));
193         pkg->command = COM_W_SEND;
194         pkg->length = 3;
195         pkgRequestAck_t *request = (pkgRequestAck_t *)pkg->data;
196         request->s.type = PKG_TYPE_REQUEST_ACK;
197         request->s.req = req;
198         request->s.ack = ack;
199         pkg->valid++;
200         uart0_send();
201 }
202
203 void notification_t::sendMessage(uint8_t idx, const char *str)
204 {
205         package_t *pkg;
206         while (!(pkg = uart0_txPackage()));
207         pkg->command = COM_W_SEND;
208         uint8_t len = strlen(str);
209         pkg->length = len + 2 + 1;
```

```
210         pkgMessage_t *message = (pkgMessage_t *)pkg->data;
211         message->s.type = PKG_TYPE_TEXT;
212         message->s.idx = idx;
213         for (uint8_t i = 0; i < len + 1; i++)
214                 message->s.str[i] = *str++;
215         pkg->valid++;
216         uart0_send();
217 }
218
219 void notification_t::sendMessageAck(uint8_t idx)
220 {
221         package_t *pkg;
222         while (!(pkg = uart0_txPackage()));
223         pkg->command = COM_W_SEND;
224         pkg->length = sizeof(pkgMessageAck_t);
225         pkgMessageAck_t *ack = (pkgMessageAck_t *)pkg->data;
226         ack->s.type = PKG_TYPE_TEXT_ACK;
227         ack->s.idx = idx;
228         pkg->valid++;
229         uart0_send();
230 }
231
232 void notification_t::displayRequest(uint8_t req)
233 {
234         tft.setBackground(Black);
235         tft.setForeground(0x667F);
236         tft.clean();
237         tft.setXY(TEXT_X, TEXT_Y);
238         tft.setZoom(TEXT_ZOOM);
239         tft.putString(PSTR("Opponent request:\n"), true);
240         tft.setX(TEXT_X);
241         tft.putString(requestName[req], true);
242
243         tft.setForeground(Black);
244         tft.setBackground(BUTTON_A_CLR);
245         tft.rectangle(BUTTON_X, BUTTON_Y, BUTTON_W, BUTTON_H, BUTTON_A_CLR);
246         tft.setXY(BUTTON_X + BUTTON_T_X(6), BUTTON_Y + BUTTON_T_Y);
247         tft.putString(PSTR("Accept"), true);
248         tft.setBackground(BUTTON_R_CLR);
249         tft.rectangle(BUTTON_X + BUTTON_W, BUTTON_Y, BUTTON_W, BUTTON_H, BUTTON_R_CLR);
250         tft.setXY(BUTTON_X + BUTTON_W + BUTTON_T_X(6), BUTTON_Y + BUTTON_T_Y);
251         tft.putString(PSTR("Reject"), true);
252 }
253
254 void notification_t::displayMessage(const char *str)
255 {
256         tft.setBackground(Black);
257         tft.setForeground(0x667F);
258         tft.clean();
259         tft.setY(TEXT_Y);
260         tft.setZoom(TEXT_ZOOM);
261         tft.putString(PSTR("Opponent message:\n"), true);
262         tft.putString(str);
263
264         tft.setForeground(Black);
265         tft.setBackground(BUTTON_TEXT_CLR);
266         tft.rectangle(BUTTON_TEXT_X, BUTTON_TEXT_Y, BUTTON_TEXT_W, BUTTON_TEXT_H, BUTTON_TEXT_CLR);
267         tft.setXY(BUTTON_TEXT_X + BUTTON_TEXT_T_X(4), BUTTON_TEXT_Y + BUTTON_TEXT_T_Y);
268         tft.putString(PSTR("Done"), true);
269 }
270
271 uint8_t notification_t::requestPool(void)
272 {
```

```
273         if (!touch.pressed())
274                 return PKG_REQUEST_RECEIVED;
275         const rTouch::coord_t pos = touch.position();
276         if (pos.y < (int16_t)BUTTON_Y || pos.x < (int16_t)BUTTON_X)
277                 return PKG_REQUEST_RECEIVED;
278         return pos.x - BUTTON_X > (int16_t)BUTTON_W ? PKG_REQUEST_REJECT : PKG_REQUEST_ACCEPT;
279 }
280
281 bool notification_t::messagePool(void)
282 {
283         if (!touch.pressed())
284                 return false;
285         const rTouch::coord_t pos = touch.position();
286         if (pos.y < (int16_t)BUTTON_TEXT_Y || pos.x < (int16_t)BUTTON_TEXT_X)
287                 return false;
288         return true;
289 }
```

Listing 7: IlMatto1/element/notification.h

```
1  /*
2   * Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #ifndef NOTIFICATION_H
6  #define NOTIFICATION_H
7
8  #include <communication.h>
9  #include <rtouch.h>
10 #include "package.h"
11
12 class notification_t
13 {
14 public:
15         void init(void);
16         package_t *pool(package_t *pkg);
17         bool show(void);
18
19         void displayRequest(uint8_t req);
20         uint8_t requestPool(void);
21         uint8_t requestAck(uint8_t req) const {return req == reqType ? reqAck : PKG_REQUEST_INVALID;}
22         void sendRequest(uint8_t req);
23         void sendRequestAck(uint8_t req, uint8_t ack);
24
25         void displayMessage(const char *str);
26         bool messagePool(void);
27         bool messageAck(uint8_t idx) const {return msgAckIdx == idx ? msgAck : false;}
28         uint8_t messageIndex(void) {return msgIdx++;}
29         void sendMessage(uint8_t idx, const char *str);
30         void sendMessageAck(uint8_t idx);
31
32 private:
33         struct request_t {
34                 uint8_t req;
35                 request_t *next;
36         } *reqs;
37
38         struct message_t {
39                 uint8_t idx;
40                 char str[BUFFER_SIZE - 1];
41                 message_t *next;
42         } *msgs;
43
44         void pushRequest(pkgRequest_t *req);
45         request_t *popRequest(void);
```

```
46          void removeRequests(uint8_t req);
47
48          void pushMessage(pkgMessage_t *msg);
49          message_t *popMessage(void);
50
51          uint16_t reqCnt, msgCnt;
52          uint8_t reqAck, reqType;
53          uint8_t msgIdx, msgAckIdx;
54          bool msgAck;
55  };
56
57  #endif
```

Listing 8: IlMatto1/element/pin.cpp

```
1   /*
2    * Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #include "common.h"
6   #include "pin.h"
7
8   #define TEXT_ZOOM     2
9   #define TEXT_X(len)   ((tft.width() - FONT_WIDTH * TEXT_ZOOM * len) / 2)
10  #define TEXT_Y        (tft.height() / 3 - FONT_HEIGHT * TEXT_ZOOM / 2)
11  #define TEXT_COLOUR   0x667F
12
13  #define PIN_ZOOM      5
14  #define PIN_X         (FRAME_X + FRAME_SIZE + FRAME_SPACE + PIN_ZOOM / 2)
15  #define PIN_Y         (tft.height() * 2 / 3 - FONT_HEIGHT * PIN_ZOOM / 2)
16  #define PIN_W         (FONT_WIDTH * PIN_ZOOM)
17  #define PIN_H         (FONT_HEIGHT * PIN_ZOOM)
18  #define PIN_SPACE     FRAME_W
19  #define PIN_COLOUR    Grey
20
21  #define FRAME_SIZE    2
22  #define FRAME_SPACE   4
23  #define FRAME_X       ((tft.width() - PIN_SPACE * PIN_LENGTH) / 2)
24  #define FRAME_Y       (PIN_Y - FRAME_SPACE - FRAME_SIZE - PIN_ZOOM / 2)
25  #define FRAME_W       (PIN_W + FRAME_SPACE * 2 + FRAME_SIZE * 2)
26  #define FRAME_H       (PIN_H + FRAME_SPACE * 2 + FRAME_SIZE * 2)
27  #define FRAME_CLR     White
28
29  uint16_t EEMEM pin_t::NVpin;
30
31  bool pin_t::init(void)
32  {
33          if (!eeprom_first()) {
34                  eeprom_read_block(&pin, &NVpin, sizeof(NVpin));
35                  return true;
36          }
37          return false;
38  }
39
40  void pin_t::set(uint16_t v)
41  {
42          pin = v;
43          eeprom_update_block(&pin, &NVpin, sizeof(NVpin));
44  }
45
46  void pin_t::display(const char *str)
47  {
48          tft.setBackground(Black);
49          tft.setForeground(TEXT_COLOUR);
50          tft.clean();
```

```
51          tft.setZoom(TEXT_ZOOM);
52
53          tft.setXY(TEXT_X(strlen_P(str)), TEXT_Y);
54          tft.putString(str, true);
55
56          uint16_t x = FRAME_X, y = FRAME_Y;
57          for (uint8_t i = 0; i < PIN_LENGTH; i++) {
58                  tft.frame(x, y, FRAME_W, FRAME_H, FRAME_SIZE, FRAME_CLR);
59                  x += FRAME_W;
60          }
61
62          tft.setZoom(PIN_ZOOM);
63          tft.setForeground(PIN_COLOUR);
64          pos = 0;
65  }
66
67  void pin_t::put(uint8_t c)
68  {
69          if (pos == PIN_LENGTH)
70                  return;
71          ip[pos] = c;
72          tft.setXY(PIN_X + PIN_SPACE * pos, PIN_Y);
73  #ifdef PIN_DISPLAY
74          tft << (c + '0');
75  #else
76          tft << '*';
77  #endif
78          pos++;
79  }
80
81  void pin_t::remove(void)
82  {
83          if (!pos)
84                  return;
85          pos--;
86          tft.rectangle(PIN_X + PIN_SPACE * pos, PIN_Y, PIN_W, PIN_H, Black);
87  }
88
89  uint16_t pin_t::input(void) const
90  {
91          uint16_t d = 0, multi = 1;
92          for (uint8_t i = PIN_LENGTH; i != 0; i--) {
93                  d += multi * ip[i - 1];
94                  multi *= 10;
95          }
96          return d;
97  }
98
99  void pin_t::restart(void)
100 {
101         while (pos)
102                 remove();
103 }
104
105 package_t *pin_t::pool(package_t *pkg)
106 {
107         uint8_t c = keypad.pool();
108         if (c == KEYPAD_NA)
109                 return pkg;
110         if (c > 9)
111                 remove();
112         else
113                 put(c);
```

```
114         return pkg;
115 }
```

Listing 9: IlMatto1/element/pin.h

```
1  #ifndef PIN_H_
2  #define PIN_H_
3
4  #include <communication.h>
5  #include <eemem.h>
6
7  #define PIN_LENGTH    4
8  //#define PIN_DISPLAY
9
10 class pin_t
11 {
12 public:
13         bool init(void);
14         package_t *pool(package_t *pkg);
15         void set(uint16_t v);
16         bool done(void) const {return pos == PIN_LENGTH;}
17         bool matched(void) const {return input() == pin;}
18         uint16_t input(void) const;
19         void restart(void);
20         void display(const char *str);
21
22 private:
23         void put(uint8_t c);
24         void remove(void);
25
26         uint8_t pos, ip[PIN_LENGTH];
27         uint16_t pin;
28         static uint16_t EEMEM NVpin;
29 };
30
31 #endif
```

Listing 10: IlMatto1/element/sketch.cpp

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #include <colours.h>
6  #include "sketch.h"
7  #include "common.h"
8
9  void sketch_t::init(void)
10 {
11         clr = White;
12         size = 1;
13         pressed = false;
14         bufferSize = PKG_SKETCH_PREPEND;
15         buffer.s.type = PKG_TYPE_SKETCH;
16
17         clean();
18 }
19
20 void sketch_t::clean(void)
21 {
22         tft.setBackground(Black);
23         tft.clean();
24         indicator::colourPicker(clr, size);
25 }
26
27 bool sketch_t::packageHandle(package_t *pkg)
```

```
{
	if (!pkg || (pkg->command != COM_W_RECV && pkg->command != COM_W_SEND))
		return false;
	pkgSketch_t *buf = (pkgSketch_t *)pkg->data;
	if (buf->s.type != PKG_TYPE_SKETCH && buf->s.type != PKG_TYPE_SKETCH_CLEAN)
		return false;
	if (buf->s.type == PKG_TYPE_SKETCH_CLEAN) {
		clean();
		return true;
	}
	uint8_t s = buf->s.size;
	uint16_t c = buf->s.clr;
	for (uint8_t i = 0; i * 4 < pkg->length - PKG_SKETCH_PREPEND; i++) {
		uint16_t x = buf->s.pos[i][0];
		uint16_t y = buf->s.pos[i][1];
		tft.rectangle(x - s / 2, y - s / 2, s, s, c);
	}
	return true;
}

void sketch_t::sendCleanPackage(void)
{
	package_t *pkg;
	uint16_t t = tick() ? tick() - 1 : TICK_CYCLE;
	while (!(pkg = uart0_txPackage()))
		if (tick() == t)
			uart0_reset();

	pkg->command = COM_W_SEND;
	pkg->length = 1;
	pkg->data[0] = PKG_TYPE_SKETCH_CLEAN;
	pkg->valid++;
	uart0_send();
}

void sketch_t::sendPackage(void)
{
	if (bufferSize == PKG_SKETCH_PREPEND)
		return;
	buffer.s.clr = clr;
	buffer.s.size = size;

	package_t *pkg;
	while (!(pkg = uart0_txPackage()));

	pkg->command = COM_W_SEND;
	pkg->length = bufferSize;
	for (uint8_t i = 0; i < bufferSize; i++)
		pkg->data[i] = buffer.d[i];
	pkg->valid++;
	uart0_send();

	bufferSize = PKG_SKETCH_PREPEND;
}

void sketch_t::writeBuffer(uint16_t x, uint16_t y)
{
	uint8_t idx = (bufferSize - PKG_SKETCH_PREPEND) / 4;

	for (uint8_t i = 0; i < idx; i++)
		if (buffer.s.pos[i][0] == x && buffer.s.pos[i][1] == y)
			return;

```

```
91          buffer.s.pos[idx][0] = x;
92          buffer.s.pos[idx][1] = y;
93          bufferSize += 4;
94  #if ((BUFFER_SIZE - PKG_SKETCH_PREPEND) % 4)
95          if (bufferSize + 4 > BUFFER_SIZE)
96  #else
97          if (bufferSize == BUFFER_SIZE)
98  #endif
99                  sendPackage();
100 }
101
102 package_t *sketch_t::pool(package_t *pkg)
103 {
104         if (shared() && packageHandle(pkg)) {
105                 uart0_done(pkg);
106                 pkg = 0;
107         }
108         rTouch::Status s = touch.status();
109         if (s != rTouch::Idle) {
110                 rTouch::coord_t pos = touch.position();
111                 if (pos.x < 0) {
112                         if (s == rTouch::Pressed)
113                                 pressed = true;
114                         else {
115                                 pressed = false;
116                                 if (keypad.colourPicker(pos, clr)) {
117                                         if (shared())
118                                                 sendPackage();
119                                         indicator::colourPicker(clr, size, false);
120                                 }
121                         }
122                         prev.x = pos.x;
123                         prev.y = pos.y;
124                 } else if (pos.x > (int16_t)(tft.width() + 10)) {
125                         sendCleanPackage();
126                         clean();
127                         bufferSize = PKG_SKETCH_PREPEND;
128                 } else {
129                         tft.rectangle(pos.x - size / 2, pos.y - size / 2, size, size, clr);
130                         if (shared())
131                                 writeBuffer(pos.x, pos.y);
132                 }
133         } else {
134                 if (shared())
135                         sendPackage();
136                 if (pressed) {
137                         pressed = false;
138                         uint8_t i = keypad.translate(keypad.keyAt(prev));
139                         if (i > 0 && i < 0x0A)
140                                 size = i;
141                         indicator::colourPicker(clr, size);
142                 }
143         }
144         return pkg;
145 }
```

Listing 11: IlMatto1/element/sketch.h

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #ifndef SKETCH_H
6  #define SKETCH_H
7
```

```
8  #include <tft.h>
9  #include <rtouch.h>
10 #include <communication.h>
11 #include "package.h"
12
13 class sketch_t
14 {
15 public:
16         void init(void);
17         package_t *pool(package_t *pkg);
18         void setShared(bool s) {sh = s;}
19         bool shared(void) const {return sh;}
20
21 private:
22         void clean(void);
23         bool packageHandle(package_t *pkg);
24         void sendPackage(void);
25         void sendCleanPackage(void);
26         void writeBuffer(uint16_t x, uint16_t y);
27
28         pkgSketch_t buffer;
29         uint8_t bufferSize;
30
31         rTouch::coord_t prev;
32         uint16_t clr;
33         uint8_t size;
34         bool pressed, sh;
35 };
36
37 #endif
```

Listing 12: IlMatto1/hw/tick.c

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #include <avr/io.h>
6  #include <avr/interrupt.h>
7  #include "tick.h"
8
9  static volatile uint16_t tk;
10
11 void tick_init(void)
12 {
13         // Clock div 8, 1ms cycle
14         TCCR1A = 0;
15         TCCR1B = _BV(WGM12) | _BV(CS11);
16         TCCR1C = 0;
17         OCR1A = 1500;
18         TIMSK1 = _BV(OCIE1A);
19         TIFR1 |= _BV(OCF1A);
20
21         tk = 0;
22 }
23
24 uint16_t tick(void)
25 {
26         return tk;
27 }
28
29 ISR(TIMER1_COMPA_vect, ISR_NOBLOCK)
30 {
31         if (++tk == TICK_CYCLE)
32                 tk = 0;
```

```
}
```

Listing 13: IlMatto1/hw/tick.h

```
/*
 *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#ifndef TICK_H
#define TICK_H

#ifdef __cplusplus
extern "C" {
#endif

#include <inttypes.h>

#define TICK_CYCLE    1000

// Using timer 1
void tick_init(void);
uint16_t tick(void);

#ifdef __cplusplus
}
#endif

#endif
```

Listing 14: IlMatto1/hw/uart0.c

```
/*
 *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "uart0.h"

// UART0 tx status masks
#define UART0_TX_SENDING     0x10   // Sending sequence of data
#define UART0_TX_STATUS      0x0F   // Status mask

// UART0 tx status
#define UART0_TX_IDLE        0                    // Idle
#define UART0_TX_WAITING     1                    // Waiting for ACK
#define UART0_TX_COMMAND     (UART0_TX_SENDING | 2) // Sending command byte
#define UART0_TX_LENGTH      (UART0_TX_SENDING | 3) // Sending length byte
#define UART0_TX_DATA        (UART0_TX_SENDING | 4) // Sending data

// UART0 rx status masks
#define UART0_RX_RECEIVING   0x10   // Receiving sequence of data
#define UART0_RX_STATUS      0x0F   // Status mask

// UART0 rx status
#define UART0_RX_IDLE        0      // Idle
#define UART0_RX_COMMAND     (UART0_RX_RECEIVING | 1)     // Receiving command byte
#define UART0_RX_LENGTH      (UART0_RX_RECEIVING | 2)     // Receiving length byte
#define UART0_RX_DATA        (UART0_RX_RECEIVING | 3)     // Receiving data

#define READ()        UDR0
#define WRITE(d)      UDR0 = d
#define ENABLE_UDREI() UCSR0B |= _BV(UDRIE0)
#define DISABLE_UDREI() UCSR0B &= ~_BV(UDRIE0)
#define ENABLE_RXCI() UCSR0B |= _BV(RXCIE0)
#define DISABLE_RXCI() UCSR0B &= ~_BV(RXCIE0)
```

```
36
37 static uint8_t ack, pendingACK;
38 volatile static uint8_t ackRecv;
39
40 static struct buffer_t {
41         struct package_t buffer[2];
42         volatile uint8_t current;
43         uint8_t status, pos;
44 } tx, rx;
45
46 void uart0_init(void)
47 {
48         // Initialise UART0_TX
49         #include <util/setbaud.h>
50         DDRD &= ~0x03;
51         PORTD |= 0x03;
52         UBRR0H = UBRRH_VALUE;
53         UBRR0L = UBRRL_VALUE;
54         UCSR0A = USE_2X << U2X0;
55         UCSR0B = (1 << RXEN0) | (1 << TXEN0);
56         UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);
57
58         // Clear flags
59         UCSR0A |= _BV(TXC0);
60         READ();
61
62         // Data struct reset
63         uart0_reset();
64
65         // Interrupt
66         ENABLE_RXCI();
67 }
68
69 void uart0_reset(void)
70 {
71         DISABLE_UDREI();
72
73         uint8_t i;
74         for (i = 0; i < 2; i++)
75                 rx.buffer[i].valid = tx.buffer[i].valid = 0;
76         tx.status = UART0_TX_IDLE;
77         rx.status = UART0_RX_IDLE;
78         rx.current = tx.current = 0;
79         rx.pos = tx.pos = 0;
80         ack = 0;
81         ackRecv = 0;
82         pendingACK = 0;
83 }
84
85 void uart0_send(void)
86 {
87         ENABLE_UDREI();
88 }
89
90 struct package_t *uart0_txPackage(void)
91 {
92         uint8_t current = tx.current;
93         struct package_t *pkg = &tx.buffer[current];
94         if (!pkg->valid)
95                 return pkg;
96         pkg = &tx.buffer[1 - current];
97         if (!pkg->valid)
98                 return pkg;
```

```
99          return 0;
100 }
101
102 uint8_t uart0_ack(void)
103 {
104         DISABLE_RXCI();
105         uint8_t ack = ackRecv;
106         ackRecv = ack ? ack - 1 : 0;
107         ENABLE_RXCI();
108         return ack;
109 }
110
111 ISR(USART0_UDRE_vect)
112 {
113         uint8_t current = tx.current;
114         struct package_t *pkg = tx.buffer + current;
115         if (tx.status & UART0_TX_SENDING)
116                 goto sending;
117
118         // ACK request?
119         while (ack) {
120                 ack--;
121                 WRITE(COM_ACK);
122         }
123
124         if (tx.status == UART0_TX_WAITING)
125                 goto disable;
126
127         // Current buffer valid?
128         if (!pkg->valid)
129                 goto disable;
130
131         // Send current buffer
132         WRITE(pkg->command);
133         if (!(pkg->command & COM_DATA))
134                 goto complete;
135         tx.status = UART0_TX_COMMAND;
136         return;
137
138 sending:
139         switch (tx.status) {
140         case UART0_TX_COMMAND:
141                 WRITE(pkg->length);
142                 if (pkg->length == 0)
143                         goto complete;
144                 tx.status = UART0_TX_LENGTH;
145                 tx.pos = 0;
146                 return;
147         case UART0_TX_LENGTH:
148         case UART0_TX_DATA:
149                 WRITE(pkg->data[tx.pos++]);
150                 if (tx.pos == pkg->length)
151                         goto complete;
152                 tx.status = UART0_TX_DATA;
153                 return;
154         }
155
156 complete:
157         tx.status = UART0_TX_WAITING; // Waiting for ACK
158         PORTB |= _BV(6);
159
160 disable:
161         DISABLE_UDREI();
```

```
}

static void uart0_received(void)
{
	uint8_t current = rx.current;
	struct package_t *pkg = rx.buffer + current;
	if (pkg->valid) {
		current = 1 - current;
		pkg = rx.buffer + current;
		if (pkg->valid)
			return;
	}
	if (pendingACK) {
		ack++;
		pendingACK--;
		ENABLE_UDREI();
	}
}

void uart0_done(struct package_t *pkg)
{
	if (!pkg)
		return;
	pkg->valid = 0;
	uart0_received();
}

struct package_t *uart0_rxPackage(void)
{
	uint8_t current = 1 - rx.current;
	struct package_t *pkg = rx.buffer + current;
	if (pkg->valid)
		return pkg;
	pkg = rx.buffer + (1 - current);
	if (pkg->valid)
		return pkg;
	return 0;
}

ISR(USART0_RX_vect)
{
	uint8_t current = rx.current;
	struct package_t *pkg = rx.buffer + current;
	if (rx.status & UART0_RX_RECEIVING)
		goto receiving;

	uint8_t c = READ();
	if (c == COM_ACK) {
		PORTB &= ~_BV(6);
		current = tx.current;
		(tx.buffer + current)->valid = 0;
		tx.current = 1 - current;
		tx.status = UART0_TX_IDLE;
		ENABLE_UDREI();
		ackRecv++;
		return;
	}

	pkg->command = c;
	if (!(c & COM_DATA))
		goto complete;
	rx.status = UART0_RX_COMMAND;
	return;
```

```
225
226 receiving:
227         c = READ();
228         switch (rx.status) {
229         case UART0_RX_COMMAND:
230                 pkg->length = c;
231                 if (pkg->length == 0)
232                         goto complete;
233                 rx.status = UART0_RX_LENGTH;
234                 rx.pos = 0;
235                 return;
236         case UART0_RX_LENGTH:
237         case UART0_RX_DATA:
238                 pkg->data[rx.pos++] = c;
239                 if (rx.pos == pkg->length)
240                         goto complete;
241                 rx.status = UART0_RX_DATA;
242                 return;
243         }
244
245 complete:
246         rx.status = UART0_RX_IDLE;
247         pkg->valid = 1;
248         rx.current = current = 1 - current;
249
250         pkg = rx.buffer + current;
251         if (!pkg->valid) {    // Can receive another package
252                 ack++;
253                 ENABLE_UDREI();
254         } else
255                 pendingACK++;
256 }
```

Listing 15: IlMatto1/hw/uart0.h

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #ifndef UART0_H
6  #define UART0_H
7
8  #ifdef __cplusplus
9  extern "C" {
10 #endif
11
12 #include <communication.h>
13
14 // Initialisation
15 void uart0_init(void);
16 // Reset tx & rx
17 void uart0_reset(void);
18
19 // Send valid tx packages
20 void uart0_send(void);
21 // Request available tx package for write, 0 for not available
22 struct package_t *uart0_txPackage(void);
23 // ACK received
24 uint8_t uart0_ack(void);
25
26 // Done with rx package, mask as free, send ACK
27 void uart0_done(struct package_t *pkg);
28 // Check available rx package for read, 0 for none
29 struct package_t *uart0_rxPackage(void);
30
```

```
#ifdef __cplusplus
}
#endif

#endif
```

Listing 16: IlMatto1/main.cpp

```
/*
 * Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <adc.h>
#include <eemem.h>
#include "menu.h"
#include "pool.h"
#include "common.h"

using namespace colours::b16;

tft_t tft;
rTouch touch(&tft);
PortraitList list(&tft);
sketch_t sketch;
keypad_t keypad;
status_t status;
pin_t pin;
notification_t notification;

void init(void)
{
	DDRB |= _BV(7);
	PORTB |= _BV(7);		// LED

	adc_init();
	adc_enable();
	uart0_init();
	tick_init();

	tft.init();
	tft.setOrient(tft.Portrait);
	tft.setForeground(0x667F);
	tft.setBackground(0x0000);
	tft.clean();

	stdout = tftout(&tft);
	notification.init();
	touch.init();
	keypad.init();
	sei();

	tft.setBGLight(true);
	touch.calibrate();
	keypad.calibrate();
	if (!pin.init())
		pool::pinSet();
	else
		pool::pinLock();
	eeprom_first_done();
}

```

```
58 int main(void)
59 {
60         init();
61 
62         tft.clean();
63         tft.setForeground(Black);
64 
65         list.refresh();
66         list.setRootItem(&menu::root::item);
67         list.display(&menu::root::item);
68 
69         for (;;)
70                 pool::list();
71 
72         return 1;
73 }
```

Listing 17: IlMatto1/menu/menu.cpp

```
1  /*
2   * Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4  
5  #include <avr/io.h>
6  #include <avr/pgmspace.h>
7  #include "menu.h"
8  
9  namespace menu
10 {
11 
12 /****************** Miscellaneous item ******************/
13 namespace misc
14 {
15         static const uint8_t PROGMEM icon_test[] = {
16                 0x83,0xC1,0x40,0x02,0x20,0x04,0x13,0xC8,0x0C,0x30,0x0C,0x30,0x92,0x49,0x91,0x89,
17                 0x91,0x89,0x92,0x49,0x0C,0x30,0x0C,0x30,0x13,0xC8,0x20,0x04,0x40,0x02,0x83,0xC1,
18         };
19 }
20 
21 namespace lock
22 {
23         static const char PROGMEM name[] = "Lock";
24         static const uint8_t PROGMEM icon[] = {
25                 0x00,0x00,0x00,0x00,0x03,0xC0,0x04,0x20,0x08,0x10,0x08,0x10,0x08,0x10,0x3F,0xFC,
26                 0x22,0xB4,0x2A,0x94,0x22,0xA4,0x2E,0xB4,0x3F,0xFC,0x3F,0xFC,0x3F,0xFC,0x3F,0xFC,
27         };
28         static listItem item = {name, icon, 0, func};
29 }
30 
31 namespace audio
32 {
33         static const char PROGMEM name[] = "Audio";
34         static const uint8_t PROGMEM icon[] = {
35                 0x01,0x00,0x03,0x10,0x05,0x08,0x09,0x24,0x11,0x12,0xE1,0x4A,0xA1,0x2A,0xA1,0x2A,
36                 0xA1,0x2A,0xA1,0x2A,0xE1,0x4A,0x11,0x12,0x09,0x24,0x05,0x08,0x03,0x10,0x01,0x00,
37         };
38         static listItem item = {name, icon, 0, func};
39 }
40 
41 namespace game
42 {
43         static const char PROGMEM name[] = "Game";
44         static const uint8_t PROGMEM icon[] = {
45                 0x38,0x41,0x44,0x22,0x82,0x14,0x82,0x08,0x82,0x14,0x44,0x22,0x78,0x41,0x00,0x00,
46                 0x00,0x00,0x10,0x7F,0x28,0x41,0x28,0x41,0x44,0x41,0x44,0x41,0x82,0x41,0xFE,0x7F,
```

```
47          };
48
49          namespace tictactoe
50          {
51                  static const char PROGMEM name[] = "Tic Tac Toe";
52                  static listItem item = {name, icon, 0, 0};
53          }
54
55          static const listItem *items[] = {&tictactoe::item, 0};
56          static listItem item = {name, icon, items, 0};
57  }
58
59  namespace sketch
60  {
61          static const char PROGMEM name[] = "Sketch";
62          static const uint8_t PROGMEM icon[] = {
63                  0x02,0x03,0x05,0x06,0x08,0x8C,0x10,0x58,0x20,0x30,0x44,0x70,0x84,0xC8,0xC7,0x84,
64                  0xA0,0x82,0x50,0xE1,0x28,0x22,0x14,0x05,0x0A,0x0A,0x05,0x14,0x02,0xA8,0x01,0x50,
65          };
66
67          namespace single
68          {
69                  static const char PROGMEM name[] = "Free play";
70                  static listItem item = {name, icon, 0, func};
71          }
72
73          namespace shared
74          {
75                  static const char PROGMEM name[] = "Shared";
76                  static listItem item = {name, icon, 0, func};
77          }
78
79          static const listItem *items[] = {&single::item, &shared::item, 0};
80          static listItem item = {name, icon, items, 0};
81  }
82
83  namespace text
84  {
85          static const char PROGMEM name[] = "Text";
86          static const uint8_t PROGMEM icon[] = {
87                  0x3F,0xFC,0x40,0x02,0x89,0x21,0x89,0x01,0x8F,0x61,0x89,0x21,0x89,0x21,0x89,0x71,
88                  0x80,0x01,0x40,0x02,0x3F,0xC4,0x00,0x22,0x00,0x12,0x00,0x09,0x00,0x05,0x00,0x02,
89          };
90          static listItem item = {name, icon, 0, func};
91  }
92
93  namespace settings
94  {
95          namespace frequency
96          {
97                  static const char PROGMEM name[] = "Frequency";
98                  static listItem item = {name, misc::icon_test, 0, func};
99          }
100
101         namespace reset_pin
102         {
103                 static const char PROGMEM name[] = "Reset PIN";
104                 static listItem item = {name, misc::icon_test, 0, func};
105         }
106
107         namespace calibration
108         {
109                 static const char PROGMEM name[] = "Calibration";
```

```
110                 static listItem item = {name, misc::icon_test, 0, func};
111         }
112
113         namespace keypadcal
114         {
115                 static const char PROGMEM name[] = "Keypad calib";
116                 static listItem item = {name, misc::icon_test, 0, func};
117         }
118
119         static const char PROGMEM name[] = "Settings";
120         static const uint8_t PROGMEM icon[] = {
121                 0x03,0xC0,0x12,0x48,0x2A,0x54,0x46,0x62,0x20,0x04,0x10,0x08,0xF1,0x8F,0x82,0x41,
122                 0x82,0x41,0xF1,0x8F,0x10,0x08,0x20,0x04,0x46,0x62,0x2A,0x54,0x12,0x48,0x03,0xC0,
123         };
124         static const listItem *items[] = {&frequency::item, &reset_pin::item, &calibration::item,
                &keypadcal::item, 0};
125         static listItem item = {name, icon, items, 0};
126 }
127
128 namespace help
129 {
130         static const char PROGMEM name[] = "Help";
131         static const uint8_t PROGMEM icon[] = {
132                 0x07,0xE0,0x0F,0xF0,0x1C,0x38,0x18,0x18,0x18,0x18,0x00,0x38,0x00,0x70,0x00,0xE0,
133                 0x01,0xC0,0x01,0x80,0x01,0x80,0x01,0x80,0x00,0x00,0x00,0x00,0x01,0x80,0x01,0x80,
134         };
135         static const listItem *items[] = {0};
136         static listItem item = {name, icon, items, 0};
137 }
138
139 namespace root
140 {
141         static const char PROGMEM name[] = "Talkie walkie 2k";
142         static const uint8_t PROGMEM icon[] = {
143                 0x00,0x00,0x07,0xE0,0x0F,0xF0,0x1F,0xF8,0x3F,0xFC,0x7F,0xFE,0x7E,0x7E,0x7C,0x3E,
144                 0x7C,0x3E,0x7E,0x7E,0x7F,0xFE,0x3F,0xFC,0x1F,0xF8,0x0F,0xF0,0x07,0xE0,0x00,0x00,
145         };
146         static const listItem *items[] = {&lock::item, &audio::item, &sketch::item, &text::item,
                &game::item, &settings::item, &help::item,
147                 0};
148         listItem item = {name, icon, items, 0};
149 }
150
151 }
```

Listing 18: IlMatto1/menu/menufunc.cpp

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #include <avr/io.h>
6  #include <avr/pgmspace.h>
7  #include "menu.h"
8  #include "pool.h"
9  #include "common.h"
10
11 using namespace menu;
12
13 bool menu::toggle::func(bool enter)
14 {
15         PINB |= _BV(7);
16         return false;
17 }
18
```

```
19 bool menu::lock::func(bool enter)
20 {
21         tft.vsNormal();
22         pool::pinLock();
23         return false;
24 }
25
26 bool menu::audio::func(bool enter)
27 {
28         tft.vsNormal();
29         if (pool::sendRequest(PKG_REQUEST_AUDIO))
30                 pool::audio();
31         return false;
32 }
33
34 bool menu::sketch::single::func(bool enter)
35 {
36         tft.vsNormal();
37         pool::sketch(false);
38         return false;
39 }
40
41 bool menu::sketch::shared::func(bool enter)
42 {
43         tft.vsNormal();
44         if (pool::sendRequest(PKG_REQUEST_SKETCH))
45                 pool::sketch(true);
46         return false;
47 }
48
49 bool menu::settings::reset_pin::func(bool enter)
50 {
51         tft.vsNormal();
52         pool::pinLock();
53         pool::pinSet();
54         return false;
55 }
56
57 bool menu::settings::calibration::func(bool enter)
58 {
59         tft.vsNormal();
60         touch.recalibrate();
61         return false;
62 }
63
64 bool menu::settings::keypadcal::func(bool enter)
65 {
66         tft.vsNormal();
67         keypad.recalibrate();
68         return false;
69 }
70
71 bool menu::settings::frequency::func(bool enter)
72 {
73         return false;
74 }
75
76 bool menu::text::func(bool enter)
77 {
78         tft.vsNormal();
79         char str[PKG_TEXT_LENGTH];
80         if (pool::textInput(PSTR("Write message:\n"), str) && *str)
81                 pool::sendMessage(str);
```

```
82          return false;
83  }
```

Listing 19: IlMatto1/menu/menu.h

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #ifndef MENU_H
6   #define MENU_H
7
8   #include <list.h>
9   #include <tft.h>
10  #include <rtouch.h>
11  #include "sketch.h"
12
13  namespace menu
14  {
15          namespace toggle {bool func(bool enter);}
16
17          namespace lock {bool func(bool enter);}
18
19          namespace audio {bool func(bool enter);}
20
21          namespace sketch
22          {
23                  namespace single {bool func(bool enter);}
24                  namespace shared {bool func(bool enter);}
25          }
26
27          namespace text {bool func(bool enter);}
28
29          namespace settings
30          {
31                  namespace frequency {bool func(bool enter);}
32                  namespace reset_pin {bool func(bool enter);}
33                  namespace calibration {bool func(bool enter);}
34                  namespace keypadcal {bool func(bool enter);}
35          }
36
37          namespace root
38          {
39                  extern listItem item;
40          }
41  }
42
43  #endif
```

Listing 20: IlMatto1/operator.cpp

```
1   // http://www.avrfreaks.net/forum/avr-c-micro-how?page=all
2
3   #include <stdlib.h>
4
5   void* operator new(size_t size)
6   {
7          return malloc(size);
8   }
9
10  void operator delete(void *ptr)
11  {
12         free(ptr);
13  }
```

Listing 21: IlMatto1/package.h

```
/*
 *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
 */

#ifndef PACKAGE_H
#define PACKAGE_H

#define PKG_TYPE_SKETCH      0
#define PKG_TYPE_SKETCH_CLEAN 1
#define PKG_TYPE_REQUEST     2
#define PKG_TYPE_REQUEST_ACK 3
#define PKG_TYPE_TEXT        4
#define PKG_TYPE_TEXT_ACK    5

#define PKG_SKETCH_PREPEND   4

#define PKG_REQUEST_SKETCH   0
#define PKG_REQUEST_AUDIO    1
#define PKG_REQUEST_COUNT    2

#define PKG_REQUEST_ACCEPT   0
#define PKG_REQUEST_REJECT   1
#define PKG_REQUEST_RECEIVED 2
#define PKG_REQUEST_CLOSED   3
#define PKG_REQUEST_INVALID  0xFF

#define PKG_TEXT_LENGTH      (BUFFER_SIZE - 2)

extern const char *requestName[PKG_REQUEST_COUNT];

union pkgSketch_t {
        uint8_t d[BUFFER_SIZE];
        struct {
                uint8_t type;
                uint8_t size;
                uint16_t clr;
                uint16_t pos[BUFFER_SIZE - PKG_SKETCH_PREPEND][2];
        } s;
};

union pkgRequest_t
{
        uint8_t d[2];
        struct {
                uint8_t type;
                uint8_t req;
        } s;
};

union pkgRequestAck_t
{
        uint8_t d[3];
        struct {
                uint8_t type;
                uint8_t req;
                uint8_t ack;
        } s;
};

union pkgMessage_t
{
        uint8_t d[BUFFER_SIZE];
        struct {
```

```
64                uint8_t type;
65                uint8_t idx;
66                char str[BUFFER_SIZE - 2];
67        } s;
68 };
69
70 union pkgMessageAck_t
71 {
72        uint8_t d[2];
73        struct {
74                uint8_t type;
75                uint8_t idx;
76        } s;
77 };
78
79 #endif
```

Listing 22: IlMatto1/pool.cpp

```
1  /*
2   *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3   */
4
5  #include "pool.h"
6  #include "common.h"
7
8  #define AUDIO_TEXT_X  ((tft.width() - FONT_WIDTH * 2 * 13) / 2)
9  #define AUDIO_TEXT_Y  (tft.height() / 3 - FONT_HEIGHT * 2 / 2)
10 #define AUDIO_SIZE    128
11 #define AUDIO_X       (tft.width() / 2 - AUDIO_SIZE / 2)
12 #define AUDIO_Y       (tft.height() * 2 / 3 - AUDIO_SIZE / 2)
13 #define AUDIO_CLR     DarkRed
14 #define AUDIO_CLR_ACT Red
15
16 #define TEXT_ZOOM     2
17
18 namespace pool
19 {
20        void pin(void);
21 }
22
23 package_t *pool::pool(void)
24 {
25        if (tick() == TICK_CLEAR)
26                while (uart0_ack());
27        return uart0_rxPackage();
28 }
29
30 void pool::pin(void)
31 {
32        uart0_done(notification.pool(::pin.pool(status.pool(pool()))));
33 }
34
35 void pool::pinLock(void)
36 {
37        ::pin.display(PSTR("Enter Passcode"));
38
39        for (;;) {
40                pin();
41
42                if (::pin.done()) {
43                        if (::pin.matched())
44                                return;
45                        else
46                                ::pin.restart();
```

```
47                }
48        }
49  }
50
51  void pool::pinSet(void)
52  {
53  restart:
54        ::pin.display(PSTR("New passcode"));
55
56        bool verify = false;
57        uint16_t v = 0;
58        for (;;) {
59                pin();
60
61                if (::pin.done()) {
62                        if (verify) {
63                                if (::pin.input() != v)
64                                        goto restart;
65                                ::pin.set(v);
66                                break;
67                        } else {
68                                v = ::pin.input();
69                                ::pin.display(PSTR("Verify passcode"));
70                                verify = true;
71                        }
72                }
73        }
74  }
75
76  void pool::sketch(bool shared)
77  {
78        ::sketch.init();
79        ::sketch.setShared(shared);
80
81        for (;;) {
82                uart0_done(notification.pool(::sketch.pool(status.pool(pool::pool()))));
83                if (shared) {
84                        if (notification.requestAck(PKG_REQUEST_SKETCH) == PKG_REQUEST_CLOSED) {
85                                indicator::refresh(Blue, PSTR("Opponent closed!"));
86                                shared = false;
87                        }
88                        status.checkRemote();
89                        indicator::checkRemote(false);
90                } else {
91                        status.checkIlMatto2();
92                        indicator::checkIlMatto2(false);
93                }
94                if (touch.pressed()) {
95                        rTouch::coord_t pos = touch.position();
96                        if (keypad.outsideLeft(pos.x + 10))
97                                break;
98                }
99        }
100       if (shared)
101               notification.sendRequestAck(PKG_REQUEST_SKETCH, PKG_REQUEST_CLOSED);
102 }
103
104 void pool::list(void)
105 {
106       for (;;) {
107               uart0_done(notification.pool(status.pool(::pool::pool())));
108               status.checkIlMatto2();
109               indicator::checkIlMatto2(true);
```

```
110                 ::list.pool(&touch);
111                 if (touch.pressed()) {
112                         rTouch::coord_t pos = touch.position();
113                         if (keypad.outsideLeft(pos.x + 10))
114                                 ::list.toUpperLevel();
115                 }
116                 while (notification.show());
117                 if (status.request.refresh) {
118                         ::list.refresh();
119                         status.request.refresh = false;
120                 }
121         }
122 }
123
124 void pool::request(uint8_t req)
125 {
126         notification.displayRequest(req);
127
128         for (;;) {
129                 uart0_done(notification.pool(status.pool(::pool::pool())));
130                 status.checkRemote();
131                 indicator::checkRemote(true);
132                 switch (notification.requestPool()) {
133                 case PKG_REQUEST_ACCEPT:
134                         goto accept;
135                 case PKG_REQUEST_REJECT:
136                         goto reject;
137                 }
138                 if (notification.requestAck(req) == PKG_REQUEST_CLOSED)
139                         return;
140                 if (touch.pressed()) {
141                         rTouch::coord_t pos = touch.position();
142                         if (keypad.outsideLeft(pos.x + 10))
143                                 goto reject;
144                 }
145         }
146 accept:
147         notification.sendRequestAck(req, PKG_REQUEST_ACCEPT);
148         switch (req) {
149         case PKG_REQUEST_SKETCH:
150                 pool::sketch(true);
151                 break;
152         case PKG_REQUEST_AUDIO:
153                 pool::audio();
154                 break;
155         }
156         return;
157 reject:
158         notification.sendRequestAck(req, PKG_REQUEST_REJECT);
159 }
160
161 bool pool::sendRequest(uint8_t req)
162 {
163         tft.setBackground(Black);
164         tft.setForeground(0x667F);
165         tft.clean();
166         tft.setZoom(2);
167         tft.putString(PSTR("Sending request...\n"), true);
168
169         uint8_t ack = PKG_REQUEST_INVALID;
170         uint16_t t = tick();
171         bool pressed = false;
172         for (;;) {
```

```
173                 if (t == tick()) {
174                         notification.sendRequest(req);
175                         if (!t--)
176                                 t = TICK_CYCLE;
177                 }
178                 uart0_done(notification.pool(status.pool(::pool::pool())));
179                 if ((ack = notification.requestAck(req)) != PKG_REQUEST_INVALID)
180                         break;
181                 if (touch.pressed())
182                         pressed = true;
183                 else if (pressed)
184                         goto close;
185         }
186
187         tft.putString(PSTR("Waiting response...\n"), true);
188         for (;;) {
189                 switch (ack) {
190                 case PKG_REQUEST_ACCEPT:
191                         return true;
192                 case PKG_REQUEST_REJECT:
193                         goto reject;
194                 }
195                 uart0_done(notification.pool(status.pool(::pool::pool())));
196                 ack = notification.requestAck(req);
197                 if (touch.pressed())
198                         pressed = true;
199                 else if (pressed)
200                         goto close;
201         }
202
203 reject:
204         tft.setForeground(Red);
205         tft.putString(PSTR("REJECTED!\n"), true);
206         for (;;) {
207                 uart0_done(notification.pool(status.pool(::pool::pool())));
208                 while (notification.show());
209                 if (status.request.refresh) {
210                         status.request.refresh = false;
211                         return false;
212                 }
213                 if (touch.pressed())
214                         pressed = true;
215                 else if (pressed)
216                         return false;
217         }
218         return false;
219
220 close:
221         notification.sendRequestAck(req, PKG_REQUEST_CLOSED);
222         return false;
223 }
224
225 void pool::audio(void)
226 {
227         tft.setBackground(Black);
228         tft.setForeground(0x667F);
229         tft.clean();
230         tft.setZoom(2);
231         tft.setXY(AUDIO_TEXT_X, AUDIO_TEXT_Y);
232         tft.putString(PSTR("Press to talk"), true);
233
234         tft.rectangle(AUDIO_X, AUDIO_Y, AUDIO_SIZE, AUDIO_SIZE, AUDIO_CLR);
235
```

```
236         bool pressed = false;
237         package_t *pkg;
238         for (;;) {
239                 uart0_done(notification.pool(status.pool(::pool::pool())));
240                 if (notification.requestAck(PKG_REQUEST_AUDIO) == PKG_REQUEST_CLOSED)
241                         goto ret;
242                 if (touch.pressed()) {
243                         rTouch::coord_t pos = touch.position();
244                         if (keypad.outsideLeft(pos.x + 10)) {
245                                 notification.sendRequestAck(PKG_REQUEST_AUDIO, PKG_REQUEST_CLOSED);
246                                 goto ret;
247                         }
248                         if (!pressed) {
249                                 tft.rectangle(AUDIO_X, AUDIO_Y, AUDIO_SIZE, AUDIO_SIZE, AUDIO_CLR_ACT);
250                                 pressed = true;
251                         }
252                         while (!(pkg = uart0_txPackage()));
253                         pkg->command = COM_W_AUDIO_TX;
254                         pkg->valid++;
255                         uart0_send();
256                 } else if (pressed) {
257                         tft.rectangle(AUDIO_X, AUDIO_Y, AUDIO_SIZE, AUDIO_SIZE, AUDIO_CLR);
258                         pressed = false;
259                         while (!(pkg = uart0_txPackage()));
260                         pkg->command = COM_W_AUDIO_TX_END;
261                         pkg->valid++;
262                         uart0_send();
263                 }
264         }
265 ret:
266         if (pressed) {
267                 while (!(pkg = uart0_txPackage()));
268                 pkg->command = COM_W_AUDIO_TX_END;
269                 pkg->valid++;
270                 uart0_send();
271         }
272 }
273
274 bool pool::textInput(const char *str, char *buf)
275 {
276         tft.setBackground(Black);
277         tft.setForeground(0x667F);
278         tft.clean();
279         tft.setZoom(TEXT_ZOOM);
280         tft.putString(str, true);
281         keypad.initText();
282         tft.setForeground(White);
283         tft.rectangle(tft.x(), tft.y(), FONT_WIDTH * TEXT_ZOOM, FONT_HEIGHT * TEXT_ZOOM,
              tft.foreground());
284
285         uint8_t len = 0;
286         for (;;) {
287                 uart0_done(notification.pool(status.pool(::pool::pool())));
288                 if (touch.pressed()) {
289                         rTouch::coord_t pos = touch.position();
290                         if (pos.x >= 0) {
291                                 *buf = 0;
292                                 return true;
293                         }
294                 }
295                 while (notification.show());
296                 char c = keypad.text();
297                 if (c == -1)
```

```
298                     continue;
299             if (c == KEYPAD_DEL) {
300                     if (!len)
301                             continue;
302                     buf--;
303                     len--;
304                     if (tft.x())
305                             tft.setX(tft.x() - FONT_WIDTH * TEXT_ZOOM);
306                     else {
307                             tft.setX(tft.width() - FONT_WIDTH * TEXT_ZOOM);
308                             tft.setY(tft.y() - FONT_HEIGHT * TEXT_ZOOM);
309                     }
310                     tft.rectangle(tft.x(), tft.y(), FONT_WIDTH * TEXT_ZOOM, FONT_HEIGHT *
                            TEXT_ZOOM, tft.background());
311             } else if (len != PKG_TEXT_LENGTH - 1) {
312                     tft << c;
313                     *buf++ = c;
314                     len++;
315                     tft.rectangle(tft.x(), tft.y(), FONT_WIDTH * TEXT_ZOOM, FONT_HEIGHT *
                            TEXT_ZOOM, tft.foreground());
316             }
317     }
318     return false;
319 }
320
321 void pool::message(const char *str)
322 {
323     notification.displayMessage(str);
324     status.request.refresh = true;
325
326     for (;;) {
327             uart0_done(notification.pool(status.pool(::pool::pool())));
328             status.checkIlMatto2();
329             indicator::checkIlMatto2(true);
330             if (notification.messagePool())
331                     break;
332             if (touch.pressed()) {
333                     rTouch::coord_t pos = touch.position();
334                     if (keypad.outsideLeft(pos.x + 10))
335                             break;
336             }
337     }
338 }
339
340 void pool::sendMessage(const char *str)
341 {
342     tft.putString(PSTR("\nSending message..."), true);
343
344     bool pressed = false;
345     uint8_t idx = notification.messageIndex();
346     uint16_t t = tick();
347     for (;;) {
348             if (t == tick()) {
349                     notification.sendMessage(idx, str);
350                     if (!t--)
351                             t = TICK_CYCLE;
352             }
353             uart0_done(notification.pool(status.pool(::pool::pool())));
354             if (notification.messageAck(idx))
355                     break;
356             if (touch.pressed())
357                     pressed = true;
358             else if (pressed)
```

```
359                     break;
360             }
361     }
```

Listing 23: IlMatto1/pool.h

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #ifndef POOL_H
6   #define POOL_H
7
8   #include <communication.h>
9
10  namespace pool
11  {
12          // Package fetch & basic pool
13          package_t *pool(void);
14
15          void pinLock(void);
16          void pinSet(void);
17          void sketch(bool shared);
18          bool textInput(const char *str, char *buf);
19          void list(void);
20          void request(uint8_t req);
21          bool sendRequest(uint8_t req);
22          void audio(void);
23          void message(const char *str);
24          void sendMessage(const char *str);
25  }
26
27  #endif
```

Listing 24: IlMatto1/status.cpp

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #include "status.h"
6   #include "common.h"
7
8   void status_t::init(void)
9   {
10          pingChk = false;
11          exist.IlMatto2 = false;
12          exist.remote = false;
13          request.refresh = false;
14  }
15
16  void status_t::ping(uint8_t cmd)
17  {
18          if (pingChk)
19                  return;
20          PINB |= _BV(7);
21          package_t *pkg = uart0_txPackage();
22          if (!pkg) {
23                  pingChk = false;
24                  return;
25          }
26          pkg->command = cmd;
27          pkg->valid++;
28          uart0_send();
29          pingChk = true;
30  }
```

```
31
32 void status_t::pingCheck(void)
33 {
34         if (!pingChk)
35                 return;
36         exist.IlMatto2Updated = true;
37         if (!(exist.IlMatto2 = uart0_ack()))
38                 uart0_reset();
39         pingChk = false;
40 }
41
42 void status_t::pingRemoteCheck(void)
43 {
44         if (!pingChk)
45                 return;
46         exist.remoteUpdated = true;
47         pingCheck();
48 }
49
50 package_t *status_t::pool(package_t *pkg)
51 {
52         if (!pkg)
53                 return 0;
54         switch (pkg->command) {
55         case COM_W_PING_TO:
56         case COM_W_PING_SU:
57                 exist.remote = pkg->command == COM_W_PING_SU;
58         case COM_PING:
59                 uart0_done(pkg);
60                 return 0;
61         case COM_W_PING: {
62                         package_t *tx = uart0_txPackage();
63                         if (tx) {
64                                 uart0_done(pkg);
65                                 tx->command = COM_W_PING_SU;
66                                 tx->valid++;
67                                 uart0_send();
68                         }
69                         return 0;
70                 }
71         }
72         return pkg;
73 }
74
75 void status_t::checkIlMatto2(void)
76 {
77         if (exist.IlMatto2Updated)
78                 exist.IlMatto2Updated = false;
79         if (tick() >= TICK_PING_CHECK)
80                 pingCheck();
81         else if (tick() >= TICK_PING)
82                 ping();
83 }
84
85 void status_t::checkRemote(void)
86 {
87         if (exist.remoteUpdated) {
88                 exist.remoteUpdated = false;
89                 exist.IlMatto2Updated = false;
90         }
91         if (tick() >= TICK_PING_REMOTE_CHECK)
92                 pingRemoteCheck();
93         else if (tick() >= TICK_PING_REMOTE)
```

```
94              ping(COM_W_PING);
95      }
```

Listing 25: IlMatto1/status.h

```
1   /*
2    *  Author: Yubo Zhi (yz39g13@soton.ac.uk)
3    */
4
5   #ifndef STATUS_H
6   #define STATUS_H
7
8   #include <communication.h>
9
10  class status_t
11  {
12  public:
13          void init(void);
14
15          void checkIlMatto2(void);
16          void checkRemote(void);
17          package_t *pool(package_t *pkg);
18
19          struct {
20                  bool IlMatto2;
21                  bool IlMatto2Updated;
22                  bool remote;
23                  bool remoteUpdated;
24          } exist;
25          struct {
26                  bool refresh;
27          } request;
28
29  private:
30          bool pingChk;
31          void ping(uint8_t cmd = COM_PING);
32          void pingCheck(void);
33          void pingRemoteCheck(void);
34  };
35
36  #endif
```

Listing 26: IlMatto2/adc.c

```
1   #include "adc.h"
2
3   void initi_ADC(void)
4   {
5           //64 division factor
6           ADCSRA |= _BV(ADPS2) | _BV(ADPS1);
7           //enable auto trigger .
8           ADCSRA |= _BV(ADATE);
9           //auto trigger source as free running mode,its initial value is zero.
10          //ADCSRB &= ~_BV(ADTS2)&~_BV(ADTS1)&~_BV(ADTS0);
11
12          //trigger source is comapre match on timer 0
13          ADCSRB |= _BV(ADTS1) | _BV(ADTS0);
14
15          //configure the reference voltage Vref to be AVcc.
16          //ADMUX |= _BV(REFS0);
17          //left adjusted result in the data registers
18          ADMUX |= _BV(ADLAR);
19          //enable ADC
20          ADCSRA |= _BV(ADEN);
21          //START CONVERSION
22          ADCSRA |= _BV(ADSC);
```

```
23          //enable ADC interrupt
24          ADCSRA |= _BV(ADIE);
25          //enable global interrupt
26          sei();
27  }
28
29  void init_adc(void)          //different sampling speed, use initi
30  {
31          ADCSRA |= _BV(ADEN) | _BV(ADATE);   //| _BV(ADPS2) | _BV(ADPS1); /* enables ADC set bits to
                set pre-scaler to /64 */
32          ADMUX |= _BV(ADLAR);
33  }
34
35  void channel_adc(uint8_t n)
36  {
37          ADMUX = n;
38  }
39
40  void adc_interrupt_enable(void)
41  {
42          ADCSRA |= _BV(ADIE);
43  }
44
45  void adc_interrupt_disable(void)
46  {
47          ADCSRA &= ~_BV(ADIE);
48  }
49
50  void adc_start(void)
51  {
52          ADCSRA |= _BV(ADSC);
53  }
54
55  void adc_stop(void)
56  {
57          ADCSRA &= ~_BV(ADSC);
58  }
```

Listing 27: IlMatto2/adc.h

```
1   #include <assert.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <avr/io.h>
5   #include <avr/interrupt.h>
6   #include <avr/pgmspace.h>
7
8   void init_adc(void);
9   void channel_adc(uint8_t n);
10  void adc_interrupt_enable(void);
11  void adc_interrupt_disable(void);
12  void adc_start(void);
13  void adc_stop(void);
14  void initi_ADC(void);
```

Listing 28: IlMatto2/pwm.h

```
1   #ifndef PWM_H
2   #define PWM_H
3
4   #include <avr/io.h>
5
6   static inline void pwm_set(const uint8_t d)
7   {
8           OCR2A = d;
9   }
```

```
10
11 static inline void pwm_init(void)
12 {
13         DDRD |= _BV(PD7);
14
15         //TCCR2A = _BV(COM2A1) | _BV(WGM20);
16         TCCR2B = 0;
17         TCNT2 = 0;
18         OCR2A = 0;
19         ASSR = 0;
20         TIMSK2 = 0;
21         TIFR2 = 0xFF;
22         TCCR2B = _BV(CS20);
23 }
24
25 static inline void pwm_enable(const uint8_t e)
26 {
27         if (e)
28                 TCCR2A = _BV(COM2A1) | _BV(WGM20) | _BV(WGM21);
29         else
30                 TCCR2A = _BV(WGM20) | _BV(WGM21);
31 }
32
33 #endif
```

Listing 29: IlMatto2/rfm12_config.h

```
1  /**** RFM 12 library for Atmel AVR Microcontrollers *******
2   *
3   * This software is free software; you can redistribute it and/or modify
4   * it under the terms of the GNU General Public License as published
5   * by the Free Software Foundation; either version 2 of the License,
6   * or (at your option) any later version.
7   *
8   * This software is distributed in the hope that it will be useful, but
9   * WITHOUT ANY WARRANTY; without even the implied warranty of
10  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
11  * General Public License for more details.
12  *
13  * You should have received a copy of the GNU General Public License
14  * along with this software; if not, write to the Free Software
15  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
16  * USA.
17  *
18  * @author Peter Fuhrmann, Hans-Gert Dahmen, Soeren Heisrath
19  */
20
21 /*****************************************************
22  *                                                   *
23  *          C O N F I G U R A T I O N                *
24  *                                                   *
25  *****************************************************/
26
27 /*
28         Connect the RFM12 to the AVR as follows:
29
30         RFM12          | AVR
31         ---------------+------------
32         SDO            | MISO
33         nIRQ           | INT0
34         FSK/DATA/nFFS  | VCC
35         DCLK/CFIL/FFIT | -
36         CLK            |  -
37         nRES           |  -
38         GND            | GND
```

```
39          ANT             |  -
40          VDD             | VCC
41          GND             | GND
42          nINT/VDI        | -
43          SDI             | MOSI
44          SCK             | SCK
45          nSEL            | Slave select pin defined below
46  */
47
48  //Pin that the RFM12's slave select is connected to
49  #define DDR_SS DDRB
50  #define PORT_SS PORTB
51  #define BIT_SS 4
52
53  //SPI port
54  #define DDR_SPI DDRB
55  #define PORT_SPI PORTB
56  #define PIN_SPI PINB
57  #define BIT_MOSI 5
58  #define BIT_MISO 6
59  #define BIT_SCK 7
60  #define BIT_SPI_SS 4
61  //this is the hardware SS pin of the AVR - it
62  //needs to be set to output for the spi-interface to work
63  //correctly, independently of the CS pin used for the RFM12
64
65  //frequency to use
66  #define FREQ 438175000UL
67  #define RFM12_BASEBAND RFM12_BAND_433
68
69  //use this for datarates >= 2700 Baud
70  #define DATARATE_VALUE RFM12_DATARATE_CALC_HIGH(115200)
71
72  //use this for 340 Baud < datarate < 2700 Baud
73  //#define DATARATE_VALUE RFM12_DATARATE_CALC_LOW(1200.0)
74
75  /**** TX BUFFER SIZE
76   */
77  #define RFM12_TX_BUFFER_SIZE 64
78
79  /**** RX BUFFER SIZE
80   * there are going to be 2 Buffers of this size
81   * (double_buffering)
82   */
83  #define RFM12_RX_BUFFER_SIZE 64
84
85  /**** INTERRUPT VECTOR
86   * define the interrupt vector settings here
87   */
88
89  //the interrupt vector
90  #define RFM12_INT_VECT (INT2_vect)
91
92  //the interrupt mask register
93  #define RFM12_INT_MSK EIMSK
94
95  //the interrupt bit in the mask register
96  #define RFM12_INT_BIT (INT2)
97
98  //the interrupt flag register
99  #define RFM12_INT_FLAG EIFR
100
101 //the interrupt bit in the flag register
```

```
#define RFM12_FLAG_BIT (INTF2)

//setup the interrupt to trigger on negative edge
#define RFM12_INT_SETUP() EICRA |= (1<<ISC21)

/**** UART DEBUGGING
 * en- or disable debugging via uart.
 */
#define RFM12_UART_DEBUG 0

/*
This is a bitmask that defines how "rude" this library behaves
       0x01: ignore other devices when sending
       0x04: don't use return values for transmission functions
*/

/* control rate, frequency, etc during runtime
 * this setting will certainly add a bit code
 **/
#define RFM12_LIVECTRL 1
#define RFM12_NORETURNS 0
#define RFM12_USE_WAKEUP_TIMER 0
#define RFM12_TRANSMIT_ONLY 0
#define RFM12_NOCOLLISIONDETECTION 1
#define RFM12_USE_POLLING 0

/* Disable interrupt vector and run purely inline. This may be useful for
 * configurations where a hardware interrupt is not available.
 */
#define RFM12_NOIRQ 0
```

Listing 30: IlMatto2/testaudioTRX_FINAL.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>
#include <ctype.h>

#include "rfm12.h"
#include "pwm.h"
#include "adc.h"
#include "uart0.h"
#include "timer.h"
#include "communication.h"

#define AUDIO 0xAA
#define IMAGE 0xCC
#define BUFFLEN 64

volatile uint8_t count = 0;
volatile uint8_t adc_buffer[BUFFLEN];
volatile uint8_t pwm_buffer[BUFFLEN];

volatile uint8_t audioTX = 0, audioRX = 0;

ISR(ADC_vect)
{
        TIFR0 |= _BV(OCF0A);  //reset compare match flag
        if (count < BUFFLEN) {
                if (!audioTX) {
                        pwm_set(pwm_buffer[count]);
                }
                adc_buffer[count] = ADCH;

```

```
33                count++;
34        }
35  }
36
37  uint8_t encryption(uint8_t data)
38  {
39        int k = 3;
40
41        uint8_t result;
42        result = ((data << k) | (data >> (8 - k)));
43        return result;
44  }
45
46  uint8_t decryption(uint8_t data)
47  {
48        int k = 3;
49
50        uint8_t result;
51        result = ((data >> k) | (data << (8 - k)));
52        return result;
53  }
54
55  int main()
56  {
57        _delay_ms(1000);
58        rfm12_init();
59        uart0_init();
60        initTimer0();
61
62        pwm_init();
63        pwm_enable(1);
64        initi_ADC();
65        adc_interrupt_enable();
66
67        sei();
68        adc_start();
69        _delay_ms(1000);
70
71        DDRC &= ~_BV(PC0);
72
73        uint8_t i;
74
75        uint8_t data_type = 0;
76        uint8_t transmit_buffer[BUFFLEN];
77        uint8_t ilmatto1_data[BUFFLEN];
78        uint8_t *receive_buffer = NULL;
79
80        struct package_t *pkg;
81
82        DDRC |= _BV(6);
83
84        while (1) {
85              pkg = uart0_rxPackage();
86
87              if (pkg) {
88                    switch (pkg->command) {
89                    case COM_PING:
90                          uart0_done(pkg);
91                          break;
92                    case COM_W_AUDIO_RX:
93                          audioRX = 1;
94                          uart0_done(pkg);
95                          break;
```

```
96                      case COM_W_AUDIO_TX:
97                              audioTX = 1;
98                              uart0_done(pkg);
99                              break;
100                     case COM_W_AUDIO_TX_END:
101                             audioTX = 0;
102                             uart0_done(pkg);
103                             break;
104                     case COM_FREQ:
105                             rfm12_set_frequency(((uint16_t) pkg->data[1] << 8) | ((uint16_t)
                                    pkg->data[0]));
106                             uart0_done(pkg);
107                             break;
108                     case COM_W_SEND:
109                             PORTC |= _BV(6);
110                             for (i = 0; i < BUFFLEN; i++) {
111                                     ilmatto1_data[i] = encryption(pkg->data[i]);
112                             }
113                             while (rfm12_tx((pkg->length), IMAGE, ilmatto1_data) ==
                                    RFM12_TX_OCCUPIED)
114                                     rfm12_tick();
115                             PORTC &= ~_BV(6);
116                             //norman wants ack from other end here
117                             uart0_done(pkg);
118                             break;
119                     default:
120                             uart0_done(pkg);
121                             break;
122
123                     }
124             }
125
126             if (audioTX) { //push button for now, can integrate the "talk" command when we add IL
                    MATTO 1
127
128                     if (count >= BUFFLEN) { //executes when adc buffer is full
129                             //prepare ADC buffer for transmit
130
131                             for (i = 0; i < BUFFLEN; i++) {
132                                     transmit_buffer[i] = encryption(adc_buffer[i]); //encrypt the data
133                             }
134
135                             while (rfm12_tx(sizeof(transmit_buffer), AUDIO, transmit_buffer) !=
                                    RFM12_TX_ENQUEUED)
136                                     rfm12_tick();
137
138                             count = 0;
139
140                     }
141             }
142             if (rfm12_rx_status() == STATUS_COMPLETE) { //executes when data has been received
143
144                     receive_buffer = rfm12_rx_buffer();
145                     data_type = rfm12_rx_type();
146
147                     if (data_type == AUDIO) {
148                             //put data on pwm_buffer
149                             for (i = 0; i < BUFFLEN; i++) {
150                                     pwm_buffer[i] = decryption(receive_buffer[i]); //encrypt
151                             }
152                             count = 0;
153
154                     } else if (data_type == IMAGE) {
```

```
                                //send data to IL MATTO 1
                                while (!(pkg = uart0_txPackage())) ;
                                pkg->command = COM_W_RECV;
                                pkg->length = rfm12_rx_len();
                                for (i = 0; i < rfm12_rx_len(); i++) {
                                        pkg->data[i] = decryption(receive_buffer[i]);
                                }
                                pkg->valid++;
                                uart0_send();
                        }
                        rfm12_rx_clear();
                }
                rfm12_tick();
        }
}
```

Listing 31: IlMatto2/timer.c

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "timer.h"

void initTimer0(void)
{
        TCCR0A = _BV(WGM01);  //CTC MODE
        TCCR0B = _BV(CS01);   // 1/8 clk

        OCR0A = 93;           //(int)(12000000/(8000*2*8) - 1);
}

void startTimer0Int(void)
{
        TIMSK0 |= _BV(OCIE0A); //interrupt on compare match A enable
}

void stopTimer0Int(void)
{
        TIMSK0 &= ~_BV(OCIE0A); //interrupt on compare match A disable
}
```

Listing 32: IlMatto2/timer.h

```
#ifndef TIMER_H
#define TIMER_H

void initTimer0(void);
void startTimer0Int(void);
void stopTimer0Int(void);
void timerTX(void);
void timerRX(void);

#endif
```

Listing 33: inc/communication.h

```
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#ifdef __cplusplus
extern "C" {
#endif

#include <inttypes.h>

// UART baudrate (750kbps)
#define BAUD          (F_CPU / 8 / (1 + 1))
```

```
12 // UART data buffer size
13 #define BUFFER_SIZE   64
14
15 // Response
16 #define COM_ACK       0
17
18 // From Il Matto 1
19
20 // Data masks
21 // Data type mask
22 #define COM_TYPE      0x0F
23 // Variable length data (length byte & data)
24 #define COM_DATA      0x80
25
26 // Data structure: Command [length data]
27 // Command without COM_DATA mask indicates no length & data
28 // Command with COM_DATA mask can have data with length <= BUFFER_SIZE
29 // Acknowledge COM_ACK after entire command sequence
30
31 // General operations
32 // No data, response {COM_ACK} indicates IlMatto2 exist
33 #define COM_PING      1
34 // Wakeup wireless module
35 #define COM_WAKEUP    2
36 // Suspend wireless module for power saving
37 #define COM_SUSPEND   3
38 // Set wireless module frequency, data 2 bytes, LE uint16_t
39 #define COM_FREQ      4
40
41 // Wireless connection operations
42 // Ping for other end
43 #define COM_W_PING    5
44 // Ping succeed reply
45 #define COM_W_PING_SU 6
46 // Ping timeout reply
47 #define COM_W_PING_TO 7
48 // Suggested timeout for remote ping, unit: ms
49 #define COM_W_PING_TIMEOUT   100
50
51 // Start transmiting audio data
52 #define COM_W_AUDIO_TX       8
53 // Start receiving audio data
54 #define COM_W_AUDIO_RX       9
55 // Stop sending & receiving audio data
56 #define COM_W_AUDIO_TX_END   10
57 // Send data to other end
58 #define COM_W_SEND           (COM_DATA | 11)
59 // Data received
60 #define COM_W_RECV           (COM_DATA | 11)
61
62 // Both send & receive can use the same buffering package type
63 struct package_t {
64        volatile uint8_t valid;
65        uint8_t command, length;
66        uint8_t data[BUFFER_SIZE];
67 };
68
69 #ifdef __cplusplus
70 }
71 #endif
72
73 #endif
```