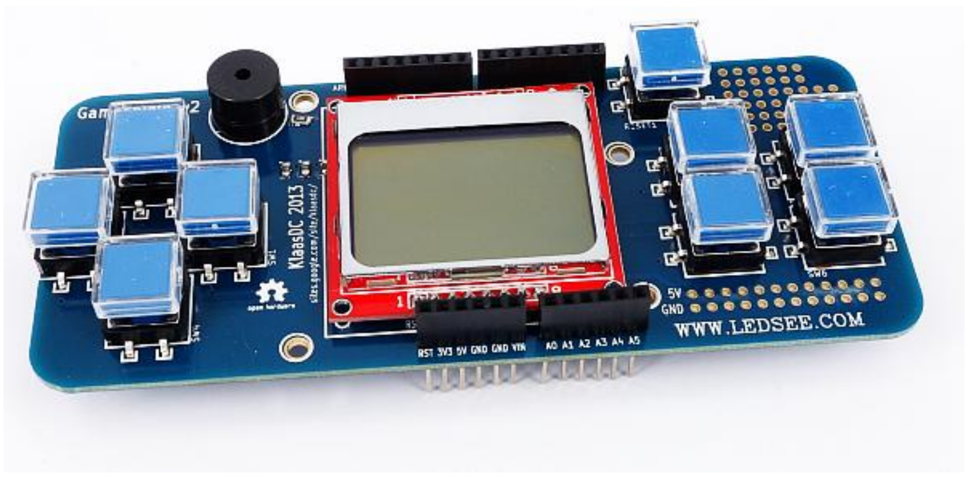
Arduino Game Shield



Klaas De Craemer – 2013

klaasdc@gmail.com

Contents

1 Component overview

The game shield is designed to plug into an Arduino Uno and transform it into a mini-Gameboy clone. The main features are:

- a Nokia 3310 monochrome LCD, with 48 x 84 pixels and a backlight consisting of 4 LEDs around the border
- 8 tactile switch buttons that can be used as inputs
- a reset button at the top
- a buzzer, that can be used as a low-quality speaker.

1.1 LCD screen

The LCD screen is the same as used in the Nokia 3310/5510 cell phones and has 48x84 pixels. Internally it has a controller chip, the PCD8544, so that it can be driven using a serial connection. On the sides there are 4 leds that function as a backlight.

Because of it's popularity, it is easy to find code examples and also a ready-made Arduino library can be found.

The controller chip works at 3.3V, which means it cannot be directly connected to the Arduino's 5V output pins. A level converter is preferred to avoid damaging the internal logic. On the Arduino game shield, this level-shifting achieved with a *Non-inverting Buffer*, type 4050. Incoming 5V data signals come out as 3.3V.

1.2 Input buttons

Next to the screen there is a small D-pad on the left and 4 general buttons on the right. These are tactile 'clicky' switches.

Because the Arduino has only a limited amount of pins available, we can save a few by using a multiplexer. With a multiplexer, you choose the button which state you want to read by setting an address. On the board is a 4051 multiplexer.

To select between the 8 buttons, a 3-bit value must be set at the address pins of the multiplexer, and then the state of the selected button appears at its output. This way we only need 4 pins of the Arduino instead of 8.

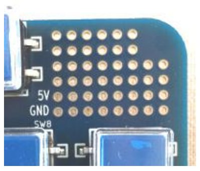
Below the D-pad there is also a reset button that is directly connected to the Arduino's reset circuit.

1.3 Buzzer

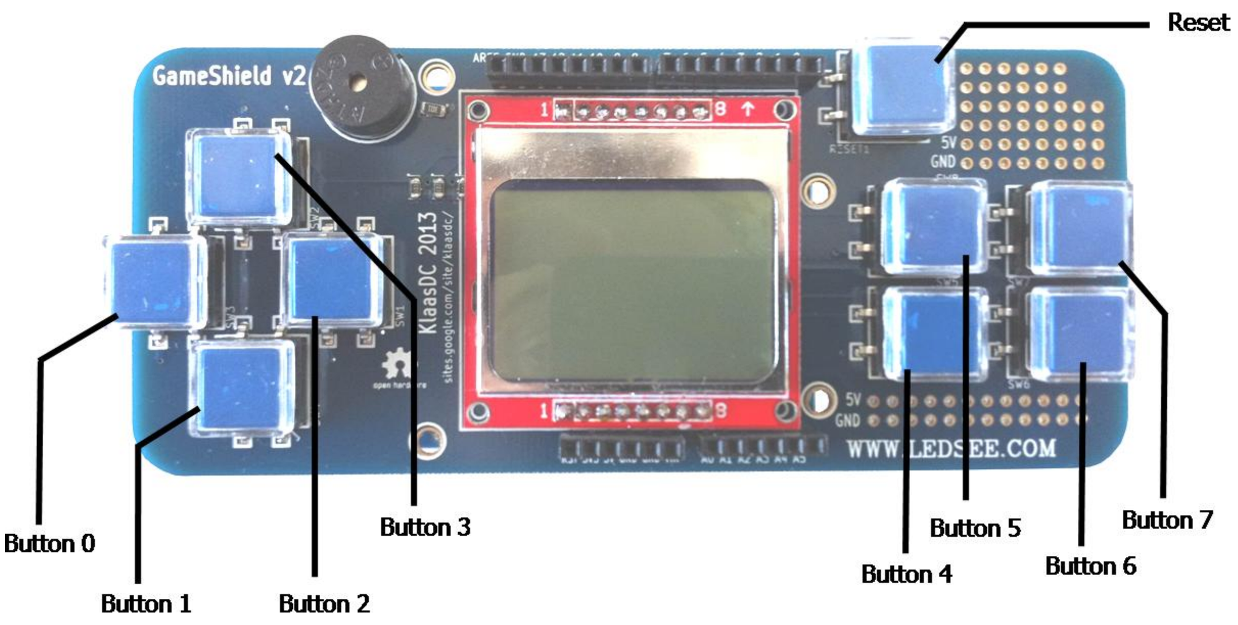
At the left top, there is space for a small buzzer. Normally, buzzers are meant to output only one specific tone. This is their resonance frequency at which maximum volume is produced, in this case about 2kHz.

When driven at other frequencies, the volume will depend on how far this frequency is from the buzzer's optimal resonance frequency. In other words, you can use it as a simple speaker, but the volume cannot be controlled other then by using a resistor.

1.4 Mini-prototyping area



At the right side of the shield, there are two small prototyping areas that expose the 5V and GND from the Arduino.



2 Testing

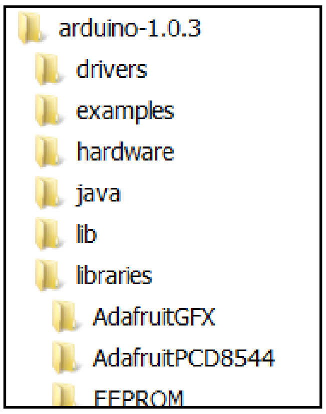
2.1 Installing the LCD libraries

As mentioned, there is an Arduino library available for the LCD screen on the shield, made by [Adafruit](). It consists of two parts, one to control the LCD, and one providing primitive drawing functions.

They are included in the zip-file for this shield:

- Adafruit-GFX-Library-master.zip
- Adafruit-PCD8544-Nokia-5110-LCD-library-master.zip

Extract the contents in the "libraries"-folder inside the Arduino-software folder so that you have this structure:

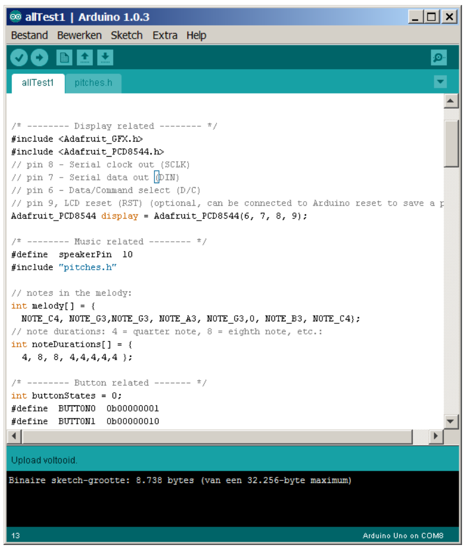


```
arduino-1.0.3
  drivers
  examples
  hardware
  java
  lib
  libraries
    AdafruitGFX
    AdafruitPCD8544
    EEPROM
```

More information on Arduino libraries can be found at http://arduino.cc/en/Guide/Libraries

2.2 Supplied examples

To test the shield, some example code is available. Open the Arduino-software and browse to the "**allTest1**" folder inside the arduino_sketches folder. This program will display an image on the screen, play a song and the wait for button inputs.



```
/* -------- Display related -------- */
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
// pin 8 - Serial clock out (SCLK)
// pin 7 - Serial data out (DIN)
// pin 6 - Data/Command select (D/C)
// pin 9, LCD reset (RST) (optional, can be connected to Arduino reset to save a p
Adafruit_PCD8544 display = Adafruit_PCD8544(6, 7, 8, 9);

/* -------- Music related -------- */
#define  speakerPin  10
#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};
// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4,4,4,4,4 };

/* -------- Button related ------- */
int buttonStates = 0;
#define  BUTTON0  0b00000001
#define  BUTTON1  0b00000010
```

```
Upload voltooid.
Binaire sketch-grootte: 8.738 bytes (van een 32.256-byte maximum)
13                                              Arduino Uno on COM8
```

You can try the other examples as well:

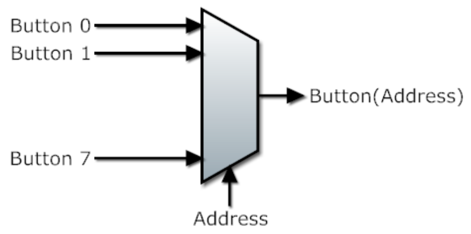
- "**buttonTest**" will loop over the button addresses and print it to the serial monitor if it is pressed. Note that, because of the pull-up resistors, a button that is pressed will result in a '0', and a non-pressed button give '1'.
- "**buttonTest2**" interprets the button address and prints text to the serial monitor accordingly. It shows how react on multiple button presses at the same time.
- "**lcdTest**" is an example that originally comes with the LCD-library and shows the possible functions that can be used: drawing rectangles, lines, text in different sizes, bitmaps ...

- "**musicTest**" is an original example for the "Tone" library and plays a few notes.
- "**musicTest2**" has some more extensive songs.
- "**snake**" is a very simple snake-game.
- "**snake2**" is a more advanced variant.
- "**spaceInv**" is a space invaders clone. Control the tank with "left"/"right" and shoot with "up".

2.3 Button reading explained

As mentioned in 1.2, the input buttons are connected to a multiplexer chip, to reduce the number of pins required to read all the buttons. You can read more about it at http://playground.arduino.cc/learning/4051

The figure below, together with the schematic in **Error! Reference source not found.** shows how it works. Pins 2,3 and 4 of the Arduino are connected to the address pin of the multiplexer. By setting a number on these pins, the corresponding button "line" is connected to pin 5 of the Arduino.



For example, if you want to read button 3, you set the address to binary value "011" and read pin 5.

In the example-code, the state of the 8 buttons is stored in a single byte value, *buttonStates*, which is updated by calling *readButtons()*:

```
void readButtons(){
  buttonStates = 0;
  int curAddress;
  for (curAddress=0; curAddress<nbOfButtons; curAddress++){
    //Turn on pins depending on current Address
    if (curAddress & 1){
      digitalWrite(addressPinA, HIGH);
    }
    else {
      digitalWrite(addressPinA, LOW);
    }
    if (curAddress & 2){
      digitalWrite(addressPinB, HIGH);
    }
    else {
      digitalWrite(addressPinB, LOW);
    }
    if (curAddress & 4){
      digitalWrite(addressPinC, HIGH);
    }
    else {
      digitalWrite(addressPinC, LOW);
    }

    //Read currently selected button input
    int button = digitalRead(buttonInputPin);
    if (button == LOW){
      buttonStates = buttonStates | (1 << curAddress);
    }
  }
}
```

As you can see, the for-loop will go over all the possible addresses (0 to 7), and update the state of the button at the current address in the *buttonStates* variable.

For more readable code, we can assign names to the numbers of the buttons:

```
#define  BUTTON0  0b00000001
#define  BUTTON1  0b00000010
#define  BUTTON2  0b00000100
#define  BUTTON3  0b00001000
#define  BUTTON4  0b00010000
#define  BUTTON5  0b00100000
#define  BUTTON6  0b01000000
#define  BUTTON7  0b10000000
```

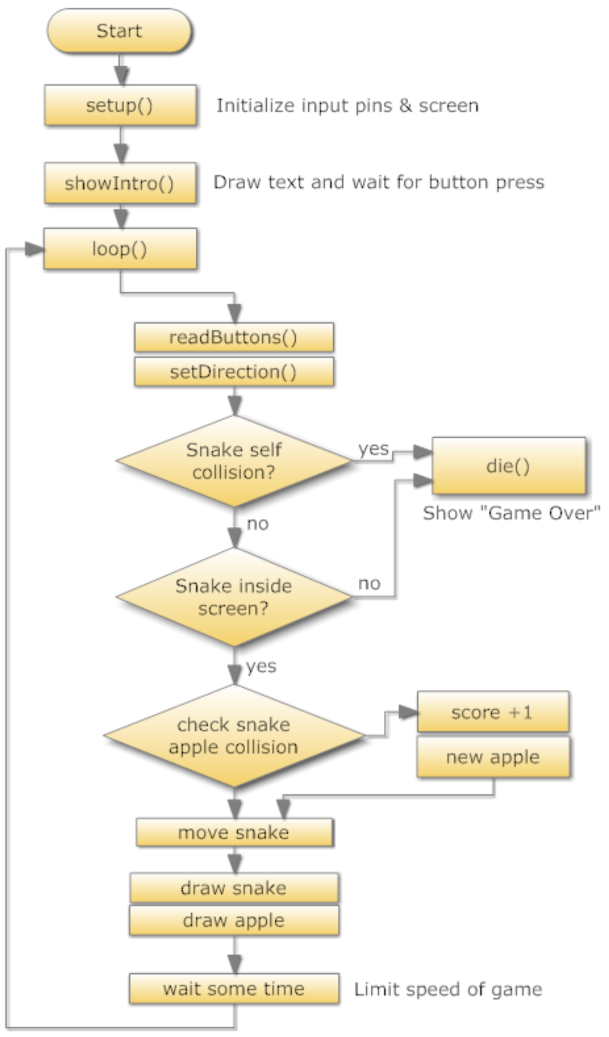
This makes the code more readable. If, after a call to readButtons, *buttonStates* differs from 0, then some button is pressed. To check each button specifically, for example for the 4th button:

```
if (buttonStates & BUTTON3){
 Serial.println("Button 3 pressed");
}
```

3 Writing games

3.1 The game loop explained

The snake example shows the main functional components that are needed in a game.



In every game there will be some kind of "game-loop":

1. First inputs (buttons/keys/...) are read.
2. The inputs are processed to update the state of the game. For example, the player's character or an enemy is moved. In this case, the snake's new direction is set.
3. Then actions are taken depending on the new state of the game. Is the player still inside the playing field? Is he hit by an enemy? Etc... In this snake game, we need to check if the snake is not colliding with itself or with the edge of the field.
   If the snake has eaten the apple, the score needs to be increased and a new apple location generated.
4. Next we prepare to draw the new game state to the screen. Move the snake one step forward in the direction that was set before.
5. Finally, we clear the screen and draw the new position of the snake (and possibly the apple).
6. Now we have to wait some time before doing another iteration of the loop. Otherwise the game would run as fast as it could. In the most simple form, this waiting can be done with a simple "delay(nbOfMilliseconds)" command.
7. The loop starts over.

3.2 Improving the loop timing

One problem with the game loop above is that the speed of the game could vary according to how much is done inside the loop.

It would be better if, instead of waiting a fixed amount, we wait long enough that every iteration of the loop takes the same amount of time. The result is a fixed frame-rate and game-speed.

Fortunately, the Arduino keeps track of the amount of milliseconds since it was reset. You can get this value by calling "millis()".

If we can compare the time when the loop was last run, and the current time, the delay can be adjusted to result in a fixed game-speed. This is implemented in the *snake2* and *spaceInv* examples:

```
//To keep the game speed constant, we keep track of the last time the snake was moved
unsigned long prevUpdateTime;

void loop() {
  ...
  //Check buttons and set snake movement direction while we are waiting to draw the next move
  unsigned long curTime;
  do{
    readButtons();
    setDirection();
    curTime = millis();
  } while ((curTime - prevUpdateTime) < 150);//Once enough time  has passed, proceed. The lower
      this number, the faster the game is
    prevUpdateTime = curTime;
}
```

Note that, instead of just doing nothing in the loop (for example by using *delay()* statements), we read and process the input buttons. This will make the game more responsive.

4 Links

http://learn.adafruit.com/nokia-5110-3310-monochrome-lcd

The home of the original LCD Arduino library

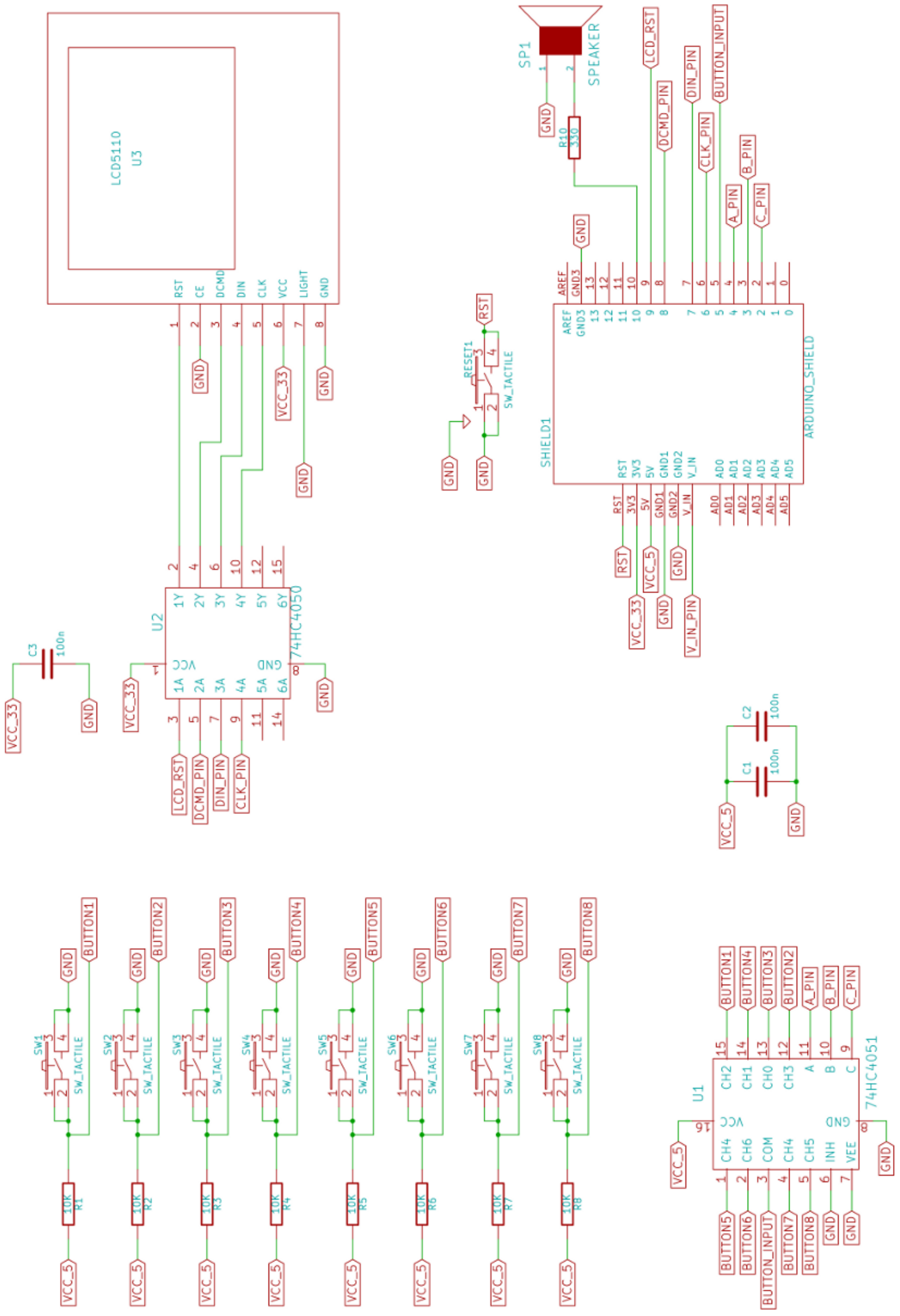
http://arduino.cc/en/Reference/SD

Makes it possible to read data from a (micro)SD card. Maybe it is possible to get graphics and music from a card.

http://code.google.com/p/arduino-playtune/

Playtune makes it possible to play music in the background, as opposed to the Tone library which waits until the end.

5 Schematic



LCD5110
U3
RST
CE
DCMD
DIN
CLK
VCC
LIGHT
GND
SP1
SPEAKER
R10
330
LCD_RST
DCMD_PIN
DIN_PIN
CLK_PIN
BUTTON_INPUT
A_PIN
B_PIN
C_PIN
SHIELD1
ARDUINO_SHIELD
RESET1
SW_TACTILE
VCC_5
VCC_33
V_IN_PIN
GND
U2
74HC4050
C3
100n
C1
C2
U1
74HC4051
BUTTON1
BUTTON2
BUTTON3
BUTTON4
BUTTON5
BUTTON6
BUTTON7
BUTTON8
10K