Arbetskrav 2 – Innbruddsalarm

Planering

Planen för innbruddsalarmet är att använda en ultrasonisk sensor för att detektera rörelse. När den triggas så har man x sekunder (satt till 10s) på sig att knappa in korrekt PIN-kod genom en keypad. Man har 3 försök på sig. Ett alarm kommer köra i gång med en lite högre frekvens (”ljusare lyd”). Om man misslyckas så måste man knappa in en annan ”master-pin” för att stänga av alarmet. Alarmet som körs i gång då ska variera mellan 2 toner. En buzzer används här för alarmet. Rätt PIN-kod stänger av alarmet.

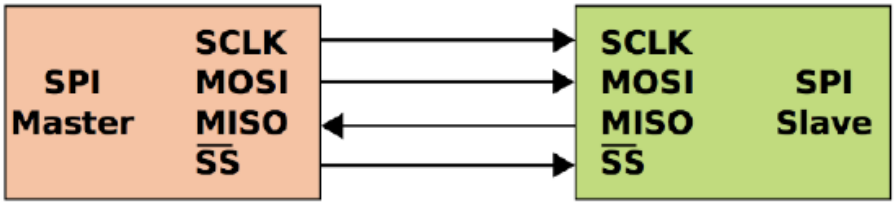
En Adafruit 1.14” 240x135 TFT LCD skärm används för att visa dagens datum och tid. Den ska också spara detta när alarmet går, för att senare kunna visa alla gånger alarmet har triggats. Eftersom den har en integrerad MicroSD-läsare så ska den också visa en bild när alarmet triggas i gång.
För att få rätt datum och tid så används en RTC-modul.
Det skulle vara kult att få till så att datum och tid när alarmen triggas sparas på microSD-kortet och då läsa/skriva från den.

Keypaden som ska användas är 4x4, där sista kolumnen har bokstäver. Om det går, så skulle det vara roligt att få till funktioner på dem. Till exemplet att ”A”-knappen gör att displayen visar alla datum och tid där alarmet har gått.

Serial Peripheral Interface

SPI (Serial Peripheral Interface) är ett så kallat synkront protokoll som används av mikrokontroller för kommunikation med andra enheter.
Den är full-duplex, dvs att data kan skicka/ta emot data samtidigt (vs. Half duplex som enbart kan göra en åt taget).
Protokollet är snabbare än I2C. Den kräver dock fler pins.
En nackdel är att den inte är bra för längre avstånd.

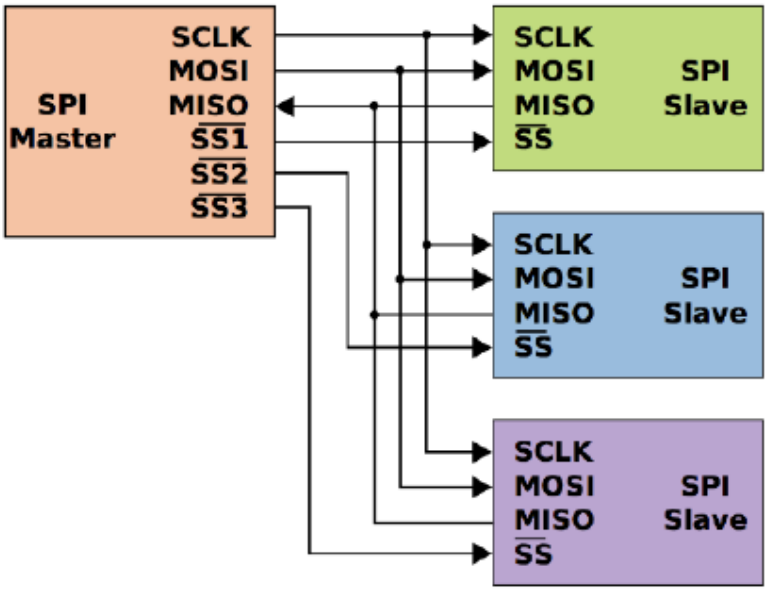
SPI använder sig av begreppen ”Master” och ”Slave”. Master är i stort sett alltid en mikrokontroller medan Slave är IC-enheten som man vill kommunicera med.
Master är alltid den som initierar en transaktion som skicka/ta emot data.



Detta görs genom 4 olika signaler:

- SCLK – Serial Clock (därav synkront), hanterar data synkroniserat med sin klocksignal.
- MOSI – Master Out Slave In, skickar data från Master till Slave.
- MISO – Master Input Slave Output, skickar data från Slave till Master
- CS/$\overline{SS}$ – Chip Select eller Slave Select, används för att välja den enhet som ska hantera kommunikationen av data.
  Sträcket ovanför SS, innebär ”Active low”. Active low för en SPI-enhet/Slave innebär att den är vald/aktiv.

Det är möjligt att koppla flera Slaves som bilden nedan. Men enbart en kan vara aktiv åt gången.



Colin M.L. Burnett, CC Share Alike 3.0 http://en.wikipedia.org/wiki/

Här är det 3st Slaves: SS1, SS2, SS3.
Om man vill kommunicera med SS1 så görs detta genom att sätta Slave Select Low till SS1. All data från SCLK, MOSI och MISO kan nu hanteras av SS1. Dataaktiviteten är dock synlig för alla Slaves, men de som inte är valda/aktiva, ignorerar dem.

SPI kräver vissa bestämda pins för att fungera på en mikrokontroller då den kräver egen hårdvara.
För en Arduino Uno så är dessa:

- SCLK – Pin 13.
- MOSI – Pin 11.
- MISO – Pin 12.
- CS/$\overline{SS}$ – Pin 10.

Komponenter brukt

- **Adafruit 240x135 1.14” IPS Screen (ST7789)** – TFT-LCD skärm från Adafruit med integrerad microSD-läsare. Använder SPI.
  Som beskrivet ovan om SPI, så används bestämda pins beroende på mikrokontroller. Namnet på de olika pins kan varierar väldigt från olika chip.
  För denna så är (vissa är samma):
  - SCLK → SCK
  - MOSI → MOSI
  - MISO → MISO
  - CS/$\overline{SS}$ → TFTCS

- **RTC ZS-042 (DS3231)** – Använder sig av I2C protokollet. Modulen är en väldigt precis variant av en RTC.
  En kristalloscillator används ofta i digitala IC för att hålla en stabil klocksignal.
  De flesta använder en extern kristall för att hålla tiden. Men beroende på temperatur så kan mätningen variera. Detta beror på att temperatur kan påverka oscillationens frekvens.
  DS3231 har en integrerad 32kHz kristalloscillator med en temperatur sensor (TCXO) för att kompensera den lilla frekvensvariationen som kan ske.
  Den har en felmarginal på ±2 minuter per år och kan hantera -40 °C till +85 °C.
  Den har också en integrerad batterihållare för att spara tiden.

  Pins:
  VCC – Power in, som klarar från 2.3V – 5.5V. Kopplas till 5V på en Arduino.
  GND – Ground/Jord.
  SDA (I2C) – Serial Data Input/Output, kopplas till Data line pin på mikrokontrollern.
  SCL (I2C) – Serial Clock Input, kopplas till Data line pin på mikrokontrollern. Används för att synkronisera data.

  På Arduino Uno så är SDA kopplad till A4 och SCL kopplad till A5, så bägge kan användas för detta.

- **Membrane switch module 4x4 (Keypad)** – Kopplingarna motsvarar antingen en rad eller en kolumn på keypaden. Detta utgör ett matrissystem.
  Första kopplingen från vänster motsvarar raden längst upp (1, 2, 3, A).
  Andra kopplingen från vänster motsvarar raden under (4, 5, 6, B), osv.
  Koppling 1–4 är rader och koppling 5–8 är kolumner.
  Koppling 5 motsvarar kolumnen längst åt vänster (1, 4, 7, *) osv.

  Man kan dock minska antalet pins från 8 till 1.
  Detta kan göras genom att mäta vilken volt som kommer igenom när man klickar på en specifik knapp. Eftersom varje kolumn är kopplad till varandra och varje rad

kopplad till varandra, så kan man exempelvis koppla vardera rad med en 1K resistor och vardera kolumn med en 220 resistor.
Exempel från min koppling:
Klickar man till exempel på:
Knapp 3 → Kopplas rad 1 med kolumn 3. Resistansen: 1k + 220 ohm = 1220 ohm.
Knapp 9 → Kopplas rad 3 med kolumn 3. Resistansen: 1k + 1k + 1k +220 + 220k.

Knapp 9: Det som sker här är att signalen skickas igenom 3 resistorer på 1k som ligger mellan varje rad. När signalen kommer till rätt rad och finner 9, så går den igenom 2st 220 resistorer för att sedan ge en output

- **Ultrasonic sensor (HC-SR04)** – Är en sensor som mäter avstånd genom lyd (40kHz frekvens). Den klarar mellan 2cm och 400cm. Under 2cm så klarar den inte mäta något och visar enbart 0. Den kopplas till 5V.
  Den har 2st sensorer:
  Trig -Fungerar som en sändare som konverterar elektrisk signal till en 40kHz frekvens.
  Echo - Fungerar som en mottagare för signalen.

  När Trig-pin sätts till HIGH så sänds en signal. Samtidigt, sätts Echo-pin till HIGH tills den mottar en signal och där direkt efter till LOW. Genom att mäta hur länge Echo-pin var HIGH, så kan man få ut avståndet.

- **Buzzer** + **330ohm resistor** – Buzzern producerar lyd genom strömmen den får. Olika frekvenser ger olika lyd. Genom att koppla till olika resistorer så kan man styra hur lågt/högt den ska låta. Här används en 330 ohm resistor för att inte lydet skulle bli för jobbigt att höra på. I verkligheten vill man säkert ha en väldigt högt alarm dock.

Processen

Först kopplade jag ihop ultrasoniska sensorn, buzzern och TFT LCD-skärmen. Dessa testade jag var för sig så att de fungerade. Sedan var det att koppla ihop dem i koden.

Jag ville få till att ultrasoniska sensorn skulle klarar av att måla avstånd under 2cm/tappad signal och då ge från sig ett lyd via buzzern.
Sedan klarar av ett specifikt avstånd för det vanliga alarmet.
Buzzern har 3 olika varianter här. En för under 2 cm och två för alarmet.

När alarmet sedan går i gång, så fick jag till att skärmen skulle visa olika ting. När alarmet först triggas, så ska den visa ett visst meddelande. När man har antingen försökt över 3 gånger att skriva in korrekt PIN-kod eller om tiden går över 10s så ska skärmen visa ett annat meddelande och spela ett annat lyd från buzzern.

Efter detta, var det dags att koppla 4x4 keypaden. Jag startade med 8 pins, men märkte senare i processen (efter jag kopplade in RTC och microSD-leser), att jag inte hade tillräckligt. Efter tips från föreläsare, så var det tydligen möjligt att göra detta med bara 1 pin.
Efter lite googling så kopplade jag om så att den enbart behövde 1 pin till Arduinon.
Före detta dock, så fick jag till PIN-kod funktionen att fungera.
En vanlig PIN-kod (4-siffrig) och en master PIN-kod (4-siffrig). Egentligen var tanken att göra master PIN-koden till 6 siffrig, men för enkelhetens skull så minska jag den till 4.

När detta var på plats så kopplade jag in RTC för att kunna vissa korrekt datum. Det var ett problem med att få korrekt datum men också efter tips från föreläsare, att använda ett batteri också, fixade problemet.
Datum och tid skrivs ut till skärmen med jämna mellanrum.

Datum och tid sparas också i en array när alarmet triggas. För att visa dessa datum så kan man klicka på knapp A, så länge inte alarmet är i gång. Det är dock en bug här som jag inte vet varför det sker. När man skriver ut dato här så skriver den ut massa random tecken också.
Knapp B tar bort allt på skärmen.

Efter att grunden var på plats, så var det dags att använda sig av ett microSD-kort. Jag började med att testa kod isolerat för att skärmen skulle kunna ladda upp en bild korrekt och sedan implementerade detta i huvudkoden. Ett problem här var då att sketch storleken gick över gränsen. Det slutade dessvärre med att jag tog bort en del error-handling kod för RTC.
Efter detta försökte jag få till read/write till microSD. Detta tog väldigt länge då jag fick en del felmeddelande från exempelkod som finns i SD-biblioteket. Efter en del strul så visade det sig inte fungera på det microSD-kortet jag använde.
Isolerat, så fungerade det sedan att spara datum och tid till microSD och även läsa från den. Men samma problematik som tidigare så fick jag problem med sketch storlek. Så detta togs bort i slutet ändå.

Det finns en del kod som är taget från exempelkod från bibliotek som refereras nedan i Referenser. Dock bara delar. Vissa delar är också modifierade för att passa in i detta alarmsystem.

Vad jag har lärt mig

Jag har lärt mig väldigt mycket om hur SPI fungerar. Det förklarade en god del generella ting också om olika pins.
Efter problematiken med kapaciteten och för att se hur snprintf_P fungerar (från exempelkod RTC) så blev det ett sidospår att läsa lite om minnehantering i Arduinon. Det var lite komplicerad men känner att jag ändå kanske fått lite mer insikt hur man kan påverka det igenom olika metoder. Exempelvis med Serial.print(F()), för att spara i flash minnet. Samt lite om PROGMEM

Sedan fick jag läsa på lite om interrupts då jag hade 2 bibliotek som försökte komma åt samma timer 2 (NewPIng och Tone). Detta gav en __vecor_7 error. Jag fick då ersätta Tone med NewTone.

Diskussion

Tidigt stötte jag på ett problem med att bibliotek försökte få tillgång till samma ting, NewPing och Tone. Detta gav en __vector_7 error då de försökte få tillgång till samma timer 2.
Jag fick då ersätta detta med NewTone istället för Tone.

Ett stort problem som jag stötte på vad att jag översteg minne-kapaciteten. Efter att jag implementerade koden för att läsa in bild från microSD-kortet så översteg jag maximala kapaciteten med 4%.
Jag försökte ändra på koden men lyckades inte nå gränsen ändå, så blev det att jag tog bort en hel del kod för felhantering av RTC:n (den här koden var taget från dess biblioteks exempel). Detta är egentligen inte så bra, men jag önskade verkligen att få till bilden. I verkligheten skulle jag hellre prioriterat korrekt datum än bilden.

Metoden som används här med att använda 1 pin för att styra keypaden genom att mäta resistansen för varje knapptryck verkar vara lite inkonsistent, troligtvis pga kopplingen, att ting kommer emot varandra. Dock enkelt att fixa genom att flytta lite här och där. Alternativt klippa av långa pinsen på resistorerna för att allt ska få plats bättre.

Vid testning för att skriva/läsa från SD-kort så var det lite problematiskt. Jag använde först en Verbatim 16GB SDHC som fungerade fint med att ladda upp .bmp till TFT-displayen. Men när jag testade SD – ReadWrite exemplet från Arduino så gick det inte.
Men när jag testade en gammal Samsung 32GB SDHC så verkade allt fungera. Väldigt osäker på varför. Bägge var formaterade till FAT32.

Isolerat så fungerar koden för att skriva datum till en .txt fil. Men det blev problematiskt i den riktiga .ino koden för alarmsystemet. Jag fick problem med plats då den översteg gränsen med ca 450 bytes. Då jag inte lyckades få ner storleken så blev det att jag inte tog med någon funktion ändå med att läsa/skriva till SD-kortet, förutom att ladda upp en bild.

Hade hade först gjort en funktion för att spara datum då alarmet triggas i en array. Den resettas ju dock vid varje start. Så det blev att jag hade kvar den i stället. Tanken var att den skulle ersättas av att läsa/skriva från SD-kortet.

Obs, vissa beskrivning som första delen av Keypaden taget från Arbetskrav 1, där samma komponent användes (dock inte sista kolumnen).

Referenser

Adafruit libraries:
Adafruit_GFX - https://www.arduino.cc/reference/en/libraries/adafruit-gfx-library/
Adafruit_ST7789 - https://github.com/adafruit/Adafruit-ST7735-Library (ST7789)
Adafruit ImageReader - https://github.com/adafruit/Adafruit_ImageReader
Adafruit_SPIFlash.h - https://github.com/adafruit/Adafruit_SPIFlash

SPI - https://github.com/arduino/ArduinoCore-avr/blob/master/libraries/SPI/src/SPI.h
SdFat - https://github.com/greiman/SdFat
Wire.h – https://github.com/esp8266/Arduino/blob/master/libraries/Wire/Wire.h
Rtc by Makuna - https://www.arduino.cc/reference/en/libraries/rtc-by-makuna/
NewPing - https://www.arduino.cc/reference/en/libraries/newping/
NewTone - https://bitbucket.org/teckel12/arduino-new-tone/wiki/Home
pitches - Från Arduino Examples - Digital – toneMelody

4x4 Keypad med en pin:
https://www.youtube.com/watch?v=G14tREsVqz0&t=206s
https://www.youtube.com/watch?v=kVw6kPSJJfw
https://www.youtube.com/watch?v=eku28glwHYo

https://bitbucket.org/teckel12/arduino-new-ping/wiki/Multiple%20Definition%20of%20%22__vector_7%22%20Error

https://microcontrollerslab.com/arduino-timer-interrupts-tutorial/