Simulerad låsmekanism

Författare: Filip Lindström

Senast ändrad: 2017-12-07
Uppdragsgivare: Mölk Utbildning AB
Kurs: Test, Verifiering och Certifiering

TINKERCAD, (https://www.myminifactory.com/pages/tinkercad)

SIMULATION, (http://www.ccsgroup.no/solidworks/programvare/simulering)

Contents

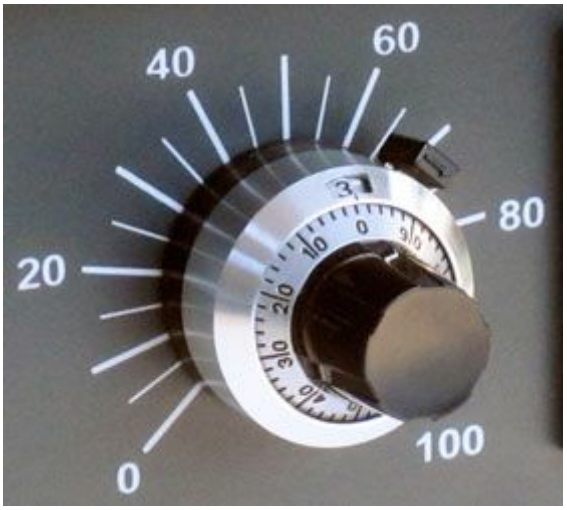
1. Inledning

I kursen ”Test, verifiering och certifiering” är ett av delmomenten att göra en simulering av ett arbete. Det här dokumentet beskrivs förloppet för simulering av ett digitalt lås.

Det digitala låset består av en Arduino Uno, fyra stycken potentiometrar, ett mikroservo, och en LED. När de fyra potentiometrarna står i specifika lägen kommer servot att röra sig och lampan att tändas, för att simulera låsmekanismen i en dörr eller kassaskåp (se figur 1.1 för illustration på hur potentiometrarna kan se ut i verklig applikation).

Hela projektet finns till förfogande på molnlösningen GitHub:

https://github.com/fille044/Simulated-Lock



Figur 1.1, Illustration

Illustration, (http://www.bourns.com/products/potentiometers/turns-counting-dials)

2. Syfte

Genom att simulera ett arbete med både input och output får man som elev en inblick i hur simulering kan användas, varför det används, och vad det kan användas till.

I projektet används en Arduino Uno på grund av att undertecknad har tidigare erfarenhet av sådan, och på så sätt kan komma igång med arbetets huvudsakliga uppgift snabbare.

Eftersom kursen diskuterar test driven utveckling kommer även detta arbete baseras på det för att få mera kunskap i ämnet.

Automatisering av byggprocessen kommer göras, även där för att utöka kunskaperna.

2.1 Frågeställning

Följande frågeställning är tänkt att besvaras under arbetets gång:

- Är det lätt att sätta sig in i en simuleringsmiljö?

- Skiljer sig simulatorn från verkligheten?

- Går det skapa testfall till en simulering?

3. Mål

Under arbetets gång ska ett projekt formuleras, planeras, skapas, testas och simuleras.

Mål med projektet är att ha en mjukvara som byggts fram genom testning, som körs på ett simulerat system i webbläsaren. Mjukvaran ska kunna köra i fler än ett simuleringsverktyg för verifiera kvalitén.

Processen att skriva mjukvara kommer inte ske i den texteditor som medföljer simuleringsverktyget, utan kommer skrivas i en installerad texteditor, versionshanteras, samt automatkompileras i flera steg, och i flera versioner.

Efter arbetets slut är målet att känna sig trygg med att använda en simulator under utvecklingen, istället för att ladda på mjukvara på faktisk hårdvara där ofta hårdvaran tar mycket tid och energi.

4. Metod

All mjukvara skrivs i Atom Editor(www.atom.io) och versionhanteras i Git och sparas på GitHub(www.github.com).

Det första som gjordes var att skapa en filstruktur där jag kunde bygga på med testfall och funktioner. Även en enklare makefil skapades med två kommandon.

4.1 Test

Testfallen som skapades var byggda på ramverket Unity(www.throwtheswitch.org/unity/) och användes för att verifiera så värdet från potentiometrarna skulle ge utslag på rätt sätt(se figur 4.1). En etta signalerar att värdet godtagits, nolla signalerar att värdet inte överensstämmer det förväntade värdet. Jag testade här värden från olika block.

```
16  void test_getInput(void)
17  {
18      TEST_ASSERT_EQUAL(17, getInput(17));
19      TEST_ASSERT_EQUAL(0, getInput(-17));
20      TEST_ASSERT_EQUAL(256, getInput(256));
21  }
22
23  void test_checkInput(void)
24  {
25      TEST_ASSERT_EQUAL(1, checkInput(200, 200));
26      TEST_ASSERT_EQUAL(1, checkInput(1000, 960));
27      TEST_ASSERT_EQUAL(0, checkInput(200, 140));
28      TEST_ASSERT_EQUAL(0, checkInput(200, 260));
29      TEST_ASSERT_EQUAL(1, checkInput(-50, 10));
30      TEST_ASSERT_EQUAL(1, checkInput(0, 0));
31  }
32
33  void test_getLock(void)
34  {
35      setLock(0);
36      TEST_ASSERT_EQUAL(0, getLock());
37      setLock(1);
38      TEST_ASSERT_EQUAL(1, getLock());
39  }
```

Figur 4.1, test_lock.c

Till en början godtogs negativa värden, något som sedan arbetades bort i funktionen *checkInput()* tills det att testet gick igenom (se figur 4.2).

```
C:\reps\Simulated-Lock>make test
Building test version
Build complete

C:\reps\Simulated-Lock>test
test_lock.c:44:test_getInput:PASS
test_lock.c:45:test_checkInput:PASS
test_lock.c:46:test_getLock:PASS

-----------------------
3 Tests 0 Failures 0 Ignored
OK
```

Figur 4.2, cmd, make test

4.2 Funktion

Funktionen *checkInput()* (se figur 4.3) är byggd med test i åtanke, varför man lätt kan välja orakel, och sedan skicka in sin simulerade input tillsammans med det förväntade svaret. Funktionen behandlar negativ input, samt om det förväntade värdet är för lågt för att hanteras. En potentiometer har svaj i signalen, och är inte precis, varför intervallet på +- 50 har valts.

```
12  int checkInput(int Input, int Answer)
13  {
14      int Min, Max;
15
16      if (Input < 0) {
17          Input = 0;
18      }
19
20      if (Answer < 50) {
21          Min = 0;
22          Max = Answer + 50;
23      }
24      else {
25          Min = Answer - 50;
26          Max = Answer + 50;
27      }
28
29      if (Input >= Min && Input <= Max)
30          return 1;
31      else {
32          return 0;
33      }
34  }
```

Figur 4.3, input.c

4.3 Makefile

För att automatisera länkning och kompilering skapade jag en ”Makefile” (se figur 4.4). I denna har jag skapat fem kommandon, *main, test, clean, install* och *help*. Jag har använt mig av en hel del variabler för att göra utbyggnation smidigare.

```
main: $(MAIN) $(FUN) $(LIB)
    @echo Building client version
    $(Q) $(C_COMPILER) -c $(FUN)
    $(Q) $(C_COMPILER) -o $(MAINTARGET) $(MAIN) $(OBJ)
    $(Q) -del -f *.o
    @echo Build complete

test: $(TESTFILES) $(FUN) $(LIB)
    @echo Building test version
    $(Q) $(C_COMPILER) -c $(FUN)
    $(Q) $(C_COMPILER) -o $(TESTTARGET) $(TESTFILES) $(OBJ) $(UNITY)
    $(Q) -del -f *.o
    @echo Build complete
```

Figur 4.4, makefile

- main: kompilerar och länkar samman funktionsfiler med huvudfilen *main.c* för att skapa den ”riktiga” exekutiven. Efter genomför kompilering raderas alla objektfiler i mappen. Utskrift talar om när byggandet startar och avslutas.
- test: har jag skapat för att kunna kompilera och länka samman en exekutiv för testning. Här inkluderar jag även Unity-biblioteket med alla dess funktioner.
  Även här raderas objektfiler, och utskrift sker både i början och slutet av byggandet.
- clean: är till för att rensa upp i mappen efter objektfiler och exekutiver.
- Install: är tänkt att vara en framtida implementation för att ladda upp kod på en enhet.
- Help: är skapad för att användaren ska kunna se de tillgängliga kommandona som finns (inspirerad av Per Böhlin) (se figur 4.5).

```
C:\reps\Simulated-Lock>make help
----  To see process, type "Q=" at the end of command  ----
*
*   main     Build client version
*   test     Build test version
*   clean    Clean away objectfiles and executives
*   install  Upload build to target
*   help     Lists available make commands
```

Figur 4.5, cmd, make help

Simulering

I simulatorverktyget Tinkercad startar man med en tom canvas, varpå man lägger till komponenter vartefter man testar. Breadboard finns att tillgå, något som enligt mig gör kopplingen smidig, tydlig och familjär.

Jag laddade upp den kod som jag skrivit tidigare till Tinkercad, bytte ut mina redan satta variabler till indata från potentiometrar, och lade till möjligheten att styra ett servo när låset skulle öppnas.

När detta var gjort startade jag simuleringen och provade mig fram med potentiometrarna (se figur 7.1). Jag hade även en lysdiod för att få en tydlig signal på om låset gått upp eller ej.

Tinkercad stödjer även seriell monitor, så möjlighet finns att skriva ut data och på så sätt felsöka koden direkt i simulatormiljön.

Även “Simulator for Arduino” (SO) från Virtronics (www.virtronics.com.au) laddades ner för att testa en annan miljö och se likheter och skillnader dem emellan. I SO valde jag min plattform(Arduino Uno) och klistrade in min kod från Tinkercad till SO:s texteditor. SO stöder inte många komponenter, utan här får man simulera mer teoretiskt. Dock väldigt användbart för att felsöka kod då man kunde stega sig igenom koden mer detaljerat.

5. Resultat

Projektet består i avslutande skede av en fungerande låsmekanism,som använder sig av fyra potentiometrar. Mjukvaran snurrar i en simulerad Arduino Uno och har testats med enhetstester på ”host”-nivå och på simulerad nivå. På host-nivå har tester gjorts i kommandotolk, och på simulerad nivå har manuella tester gjorts.

När de fyra potentiometrarna står i sina rätta lägen (300 – 200 – 1000 – 750) kommer ett servo arbeta och en lysdiod kommer tändas för att signalera att låsmekanismen har gått från stängd till öppen. När någon av potentiometrarna sedan ändrar sitt läge kommer servot att stänga och lysdioden kommer slockna. Som koden är skriven nu är potentiometerns arbetsområde uppdelat i 10, så med de fyra potentiometrarna (10*4) finns 40 olika kombinationer av lägen. Man kan lätt ställa ner arbetsområdena till 20 per enhet genom att ändra parameter SENSITIVITY från 50 till 25, och på så sätt få 80 olika kombinationer.

Simuleringen är gjord i två simuleringsverktyg, Tinkercad(www.tinkercad.com) samt i Simulator for Arduino från Virtronics(www.virtronics.com.au).

Projektet har hela tiden drivits med TDD i åtanke. Detta tillsammans med den restrikterade inmatningen från användaren i form av dyra vred gör att systemet är relativt säkert och beprövat.

6. Diskussion och reflektion

6.1 Tankar kring simulering

Det här arbetet har givit mig en inblick i varför man skulle använda sig av en simulator i en utvecklingsprocess, vad man kan använda den till, och främst hur man drar nytta av den.

Projektet har varit relativt avskalat, och det med flit. Dels för att jag ville kunna slutföra det, även om det bara är i simuleringssyfte, men också för att jag ville göra hela kedjan från idé, till att skapa testfall, till att skapa funktioner efter testfallen och sedan implementera dom i simulatorn. Även automatisering gjordes under tiden utvecklingen pågick för att spara tid i den delen och för att slipa på kunskaperna till framtiden(läs ex-jobb).

6.2 Tinkercad

Den första tanken som slog mig när jag började använda Tinkercad, var hur lätt det var att navigera sig runt i miljön, och hur lätt man byggde upp en system i simulatorn. Texteditorn som finns i verktyget fungerar säkert för nybörjare som vill blinka en LED, men för något mer är det är den klumpig och otydlig. Färgsättningen, indenteringar och det faktum att texteditorn måste minimeras för att kunna köra simuleringen på ett vettigt sätt gjorde att jag tog beslutet att göra all min utveckling i externa program. När mjukvaran lirar på host-nivå och testerna gått igenom, så flyttas denna helt enkelt över till simulatorns editor.

Prestandan i de flesta gratis simulatorer är väldigt begränsad, och bristen på tangentbords-genvägar gjorde det helt enkelt osmidigt att sköta utvecklingen därifrån. Jag lyckades inte heller skapa flera filer i simulatorn, något som är ett måste för att jag ska känna mig organiserad och hemma. När simulering skulle köras fick jag plocka funktionerna från respektive fil och addera dom till simulatorns editor.

6.3 Liknelser och avvikelser

Jag har tidigare använt mig av Fritzing för att rita kretsscheman, jag har programmerat Arduino mycket i deras utvecklingsmiljö, samt pysslat mycket med hårdvaran kring det. Med det sagt tyckte jag inte att steget in i Tinkercad var speciellt stort, utan jag kände mig bekväm i det direkt, och det enda detta egentligen lägger till utöver de andra programmen är möjligheten att starta simuleringen.

Just Tinkercad påminner väldigt mycket om Fritzing i sättet att arbeta och navigera, lite som en nedskalad webbversion av Fritzing. Fullt tillräcklig för att simulera Arduino som troligtvis räcker till de flesta arbeten man kommer göra framöver. I mitt fall är oftast hårdvaran det som tar mest tid, så att kunna använda sig av ett sådant här verktyg kommer garanterat spara mig många timmar och tårar i framtiden.

6.4 Övriga simulatorer

VBB4Arduino laddades ner och startades, dock krävs licens för att kunna kompilera och simulera i det programmet. Jag provade att starta ett projekt ändå, och det kändes mycket mera polerat än Tinkercad, med en mycket smidigare och mer mångsidig utvecklingsmiljö.

Man kunde se en filstruktur, så troligtvis kan man programmera i flera filer, vilket jag känner mig mer bekväm att arbeta i.

Att bygga upp simulatorn var något krångligare än Tinkercad, och mera lik Fritzing, alltså mer avancerad.

De flesta mjukvaror som verkar vara mer kompletta är tyvärr av betaltyper, där vissa har prövotider på sina mjukvaror. Jag är dock inte beredd att använda upp prövotiden på det här projektet, utan sparar den till ex-jobbet ifall det skulle bli aktuellt att simulera Arduino då.

6.5 Svar på frågeställning

- Är det lätt att sätta sig in i en simuleringsmiljö?

Svaret är delat, och beror helt och hållet på vilken miljö som används och om man har tidigare erfarenhet. Jag tror dock att miljöer som Tinkercad kan lämpa sig väl till nybörjare som vill känna på tekniken men inte är beredd att köpa egen hårdvara.

Program som "Simulator for Arduino" från Virtronics(www.virtonics.com.au) är inte lika lätta att använda. Användargränssnittet är rörigt, och kräver tillvänjning för att kunna navigeras på ett vettigt sätt. Bortsett från det erbjuder programmet noggrann avläsning av processer, samt möjligheten att se variablers tillstånd vid varje steg i koden. Man kan här alltså stega sig fram rad för rad, vilket öppnar upp för bra möjligheter när det kommer till felsökning.

- Skiljer sig simulatorn från verkligheten?

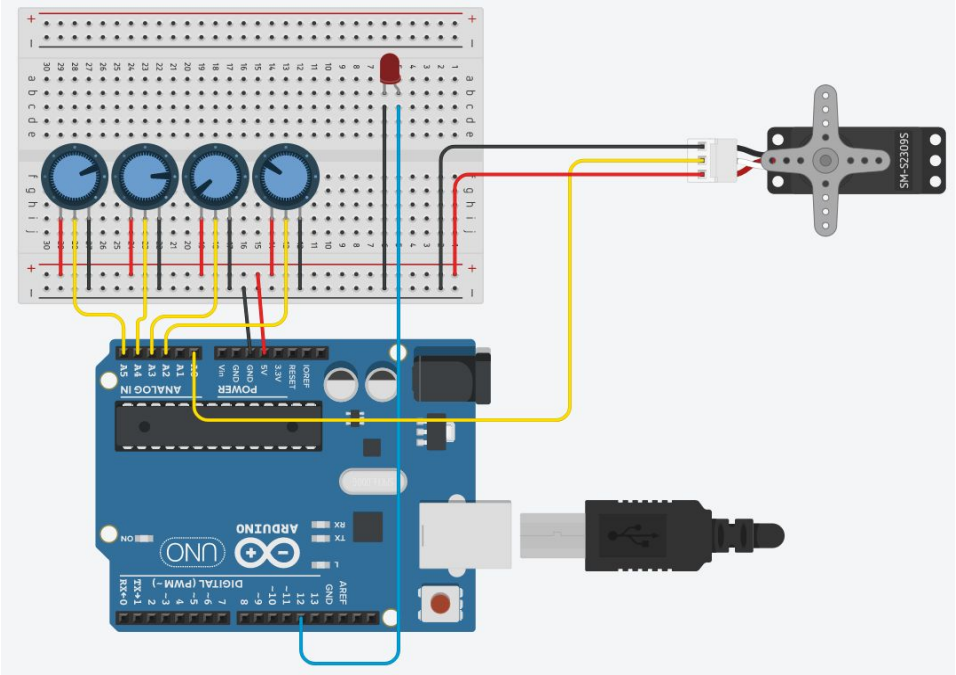
Simulation i ett program som Tinkercad "förfinar" verkligheten, där allt går väldigt smidigt och hårdvarans alla kopplingar är lättöverskådliga. Den som byggt något på en breadboard med jumper-kablar vet att verkligheten ter sig annorlunda.

In- och utgångar på en verklig Arduino har en tendens till att spela en ett spratt då och då, i form av brusiga signaler och känsliga kopplingar. I de simulatorer jag känt på finns inte någon sådan "verklighet" utan allt fungerar som det teoretiskt sett borde.

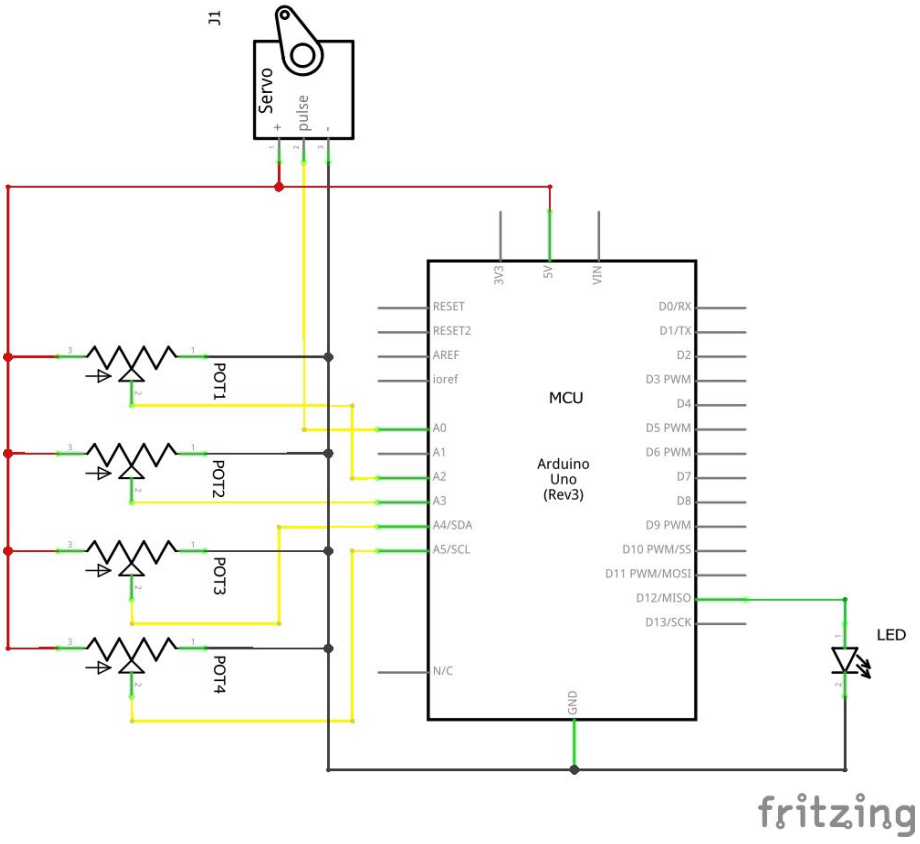
- Går det skapa testfall till en simulering?

Frågan fick jag besvarad relativt tidigt i projektet, där jag skrev testfall på host-nivå till de funktioner som sedan implementeras i simulatorn med små justeringar för att passa i den miljön.

7. Bilaga kopplingsschema och fritzing-skiss



Figur 7.1, Skiss från simulator Tinkercad



Figur 7.2, Kretsschema ritat i Fritzing

8. Källkod

```
#include <Servo.h>
#define DEBUG 1

Servo servo1;

/* ----------------------------------------*/
void serialDebug(void)
{
    Serial.print("A5 ");
    Serial.println(analogRead(A5));
    Serial.print("A4 ");
    Serial.println(analogRead(A4));
    Serial.print("A3 ");
    Serial.println(analogRead(A3));
    Serial.print("A2 ");
    Serial.println(analogRead(A2));
}


/* ----------------------------------------*/
int checkInput(int Input, int Answer)
{
    int Min, Max;

    if (Input < 0) {
        Input = 0;
    }

    if (Answer < 50) {
        Min = 0;
        Max = Answer + 50;
    }
    else {
        Min = Answer - 50;
        Max = Answer + 50;
    }

    if (Input >= Min && Input <= Max)
        return 1;
    else {
        return 0;
    }
```

```
}
/* ----------------------------------------*/
void setLock(int State)
{
    if (State == 1) servo1.write(180);
    else if (State == 0) servo1.write(0);
}

/* ----------------------------------------*/
void setup()
{
    servo1.attach(A0);
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    Serial.begin(9600);
}


/* ----------------------------------------*/
void loop()
{
    if (DEBUG == 1) serialDebug();

    if ((checkInput(analogRead(A5), 300) &&
         checkInput(analogRead(A4), 200) &&
         checkInput(analogRead(A3), 1000) &&
         checkInput(analogRead(A2), 750)) == 1) {
         digitalWrite(12, HIGH);
       setLock(1);
       Serial.println("HIGH");
    }
    else {
        digitalWrite(12, LOW);
        setLock(0);
        Serial.println("LOW");
    }
    delay(1000);
}
```

9. Någonting som visar en körning av simuleringen

Tinkercad Simulation of pot-lock (https://youtu.be/ZGZyfQS7aPE)