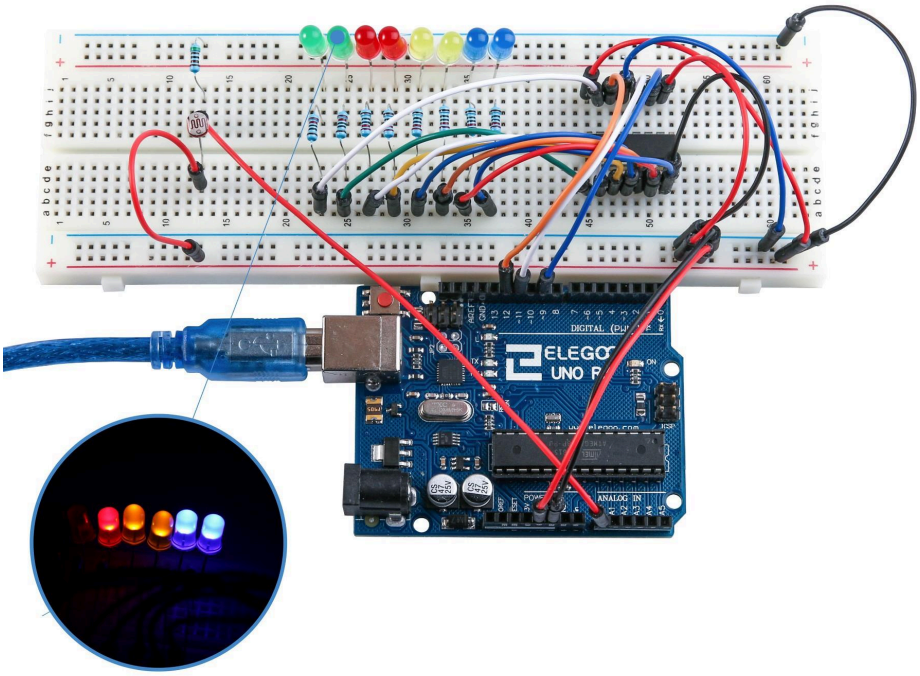
MODUL 5

Integration · Hackathon

Alla fyra modulerna, en krets, ett fungerande tjuvlarm.



Den typ av sammansatt krets ni byggde på hackathonen — RGB-LED:er, fotocell, buzzer, sammanvävda på en enda breadboard. Här en variant där fotocellen styr LED:erna direkt.

Vad du gjorde i dag

Sista träffen är en sammansmältning. Ingen ny teori, ingen ny syntax — allt ni behövde kunde ni redan. I stället satte ni ihop bitarna ni byggt separat och gjorde ett fungerande system av dem.

Systemet ni byggde:

- **En knapp** (Modul 3) som togglar ett ”larm-läge”.
- **En tilt-sensor** (Modul 4) som upptäcker om larmet rörs.
- **En fotocell** (Modul 4) som mäter omgivningsljus.
- **En RGB-LED** (Modul 2) som lyser stämningsljus i mörker.
- **En buzzer** (Modul 3) som tjuter vid tilt-event.
- **Ett program** som binder ihop alltihop.

Reglerna var enkla, men programmet som realiserade dem krävde att varje bit från de föregående fyra modulerna sammanvävdes till en enda loop:

1. Knappen togglar `larmPaslaget` med **edge-detection**.
2. Om `larmPaslaget && tilt-sensorn rörs` → **tjut + rött blink**.
3. Om `!larmPaslaget && fotocellen är mörk` → **stämningsljus**.
4. Annars → **tyst och mörkt**.

Det är i princip ett minimalt embedded-system. Ni har byggt ett sådant. Grattis — ni kan nu beskriva grunderna för allt från disktermometrar till enkla RC-bilar i samma ordlista.

Systemets arkitektur

PIN-TILLDELNING

Samma pinnar som i varje lektion — återanvänds direkt:

`knappPin`	D9 — från Modul 3
`tiltPin`	D2 — från Modul 4
`ldrPin`	A0 — från Modul 4
`buzzerPin`	D12 — från Modul 3
`ledR`	D6 — från Modul 2
`ledG`	D5 — från Modul 2
`ledB`	D3 — från Modul 2

Alla dessa pinnar är PWM (`~`-markerade) för R/G/B-kanalerna så att ni kan variera ljusstyrkan. Digital2 och Digital12 är vanliga digital-pinnar — inga krav på PWM där.

SENSE-ACT-LOOPEN

Strukturen för hela programmet är densamma som alla embedded-system någonsin skrivna:

1. **Läs av omvärlden** — fotocell, tilt, knapp.
2. **Uppdatera internt tillstånd** — hantera edge-detection för knappen.
3. **Bestäm vad som ska hända** — if/else baserat på `larmPaslaget` och mätvärdena.
4. **Styr utvärlden** — skriv till RGB-LED och buzzer.
5. **Paus** — kort `delay` för att inte bränna CPU:n.
6. Börja om.

Nedan bygger vi upp den här loopen **i tre steg**, precis som ni själva ska göra under hackathonen. Den fullständiga sammansatta sketch:en ligger i Bilaga D, som ni får efter hackathonen — kursledaren har den om ni kör fast helt och behöver titta på en referens.

Bygg inkrementellt

Den vanligaste nybörjarfällan är att försöka skriva hela programmet direkt. Bygg **ett steg i taget** och bekräfta att varje steg fungerar innan ni går vidare. När något går sönder vet ni exakt vad det är — det är det ni nyligen la till.

STEG 1 — KNAPPEN TOGGLAR LARM-LÄGET

Börja med att bara läsa knappen med edge-detection och printa till Serial Monitor varje gång `larmPaslaget` ändras. Inga LED:ar, ingen buzzer, ingen sensor. Bara: ”fungerar togglet-ringen?”

```
const int knappPin = 9;
bool larmPaslaget = false;
int  lastKnappState = HIGH;

void setup() {
  Serial.begin(9600);
  pinMode(knappPin, INPUT_PULLUP);
  Serial.println("Start. Larm av.");
}

void loop() {
  int knappState = digitalRead(knappPin);
  if (knappState == LOW && lastKnappState == HIGH) {
    larmPaslaget = !larmPaslaget;
    Serial.print("Larm nu ");
    Serial.println(larmPaslaget ? "PÅ" : "AV");
  }
  lastKnappState = knappState;
  delay(10);
}
```

Ladda upp. Tryck. Serial Monitor ska skriva ”Larm nu PÅ” och ”Larm nu AV” omväxlande — en rad per tryck. **Om den flippar flera gånger per tryck har du en bug. Fixa först. Gå inte vidare.**

STEG 2 — LÄGG TILL TILT-SENSOR OCH BUZZER

Utöka: när larmet är på **och** tilten lutas → buzzern tjuter. När larmet är av eller tilten står upprätt → tyst.

```
// Lägg till överst:
const int tiltPin   = 2;
const int buzzerPin = 12;

// I setup(), efter pinMode för knappen:
pinMode(tiltPin,   INPUT_PULLUP);
pinMode(buzzerPin, OUTPUT);

// I loop(), efter edge-detection-blocket men före delay:
bool tiltLutad = (digitalRead(tiltPin) == LOW);
if (larmPaslaget && tiltLutad) {
  digitalWrite(buzzerPin, HIGH);
} else {
  digitalWrite(buzzerPin, LOW);
}
```

Ladda upp. Toggla larmet på med knappen. Luta tilten → det ska tjuta. Återställ larmet till av → tyst även om tilten lutas. Bekräfta båda innan nästa steg.

STEG 3 — LÄGG TILL STÄMNINGSLJUS I MÖRKER

Sista biten: när larmet är **av** och fotocellen läser under en tröskel → RGB-LED:en tänds mjukt. Annars mörk.

```
// Lägg till överst:
const int ldrPin       = A0;
const int ledR = 6, ledG = 5, ledB = 3;
const int morkTroskel  = 300;     // kalibrera själv

// I setup(), efter buzzerPin-raden:
pinMode(ledR, OUTPUT);
pinMode(ledG, OUTPUT);
pinMode(ledB, OUTPUT);

// I loop(), utöka den if/else som styr buzzern till tre grenar:
int ljus = analogRead(ldrPin);
if (larmPaslaget && tiltLutad) {
  digitalWrite(buzzerPin, HIGH);
  analogWrite(ledR, 255); analogWrite(ledG, 0); analogWrite(ledB, 0);
} else if (!larmPaslaget && ljus < morkTroskel) {
  digitalWrite(buzzerPin, LOW);
  analogWrite(ledR, 120); analogWrite(ledG, 60); analogWrite(ledB, 20);
} else {
```

```
  digitalWrite(buzzerPin, LOW);
  analogWrite(ledR, 0); analogWrite(ledG, 0); analogWrite(ledB, 0);
}
```

Ladda upp. Täck fotocellen (larmet av) → LED lyser varmt. Toggla larmet på och luta → tjut + rött. Testa alla kombinationer.

Den färdiga, rensade och kommenterade sammansatta sketch:en med en hjälpfunktion för att sätta färg: **Bilaga D** (delas ut efter hackathonen).

Tips för hackathon-formatet

SERIAL MONITOR ÄR INTE VALFRI

Varje gång du lägger till ett nytt villkor eller en ny variabel, printa den. Printa `larmPaslaget`. Printa `ljus`. Printa `state`. Printa varje gren av varje if/else. Om beteendet är mystiskt — läs vad Serial Monitor säger, inte vad du **tror** står i koden.

> **DEBUG-PRINT SNABBRECEPT**
>
> ```
> Serial.print("larm=");
> Serial.print(larmPaslaget);
> Serial.print(" ljus=");
> Serial.print(ljus);
> Serial.print(" tilt=");
> Serial.println(tilt);
> ```
>
> Kör det på slutet av varje loop (efter ett lämpligt `delay(200)`) så har du en kontinuerlig ström av sanning.

EDGE-DETECTION ÄR ICKE FÖRHANDLINGSBAR

För knappen: **alltid** edge-detection, aldrig direkt `if (digitalRead(knappPin) == LOW)`. Utan edge-detection hoppar `larmPaslaget` fram och tillbaka hundratals gånger per tryckning, och du har ingen aning om vilket värde den slutade på när du släppte.

För tilten: edge-detection är mer valfritt. Ni kan agera direkt på ”tilt är LOW nu”, och det fungerar oftast bra. Men samma problem med studs kan uppstå — lägg `delay(50)` efter läsningen.

ORDNA KODEN I BLOCK

När stegen 1–3 ovan är hopfogade ser loopen typiskt ut så här:

1. **Läs inputs** — `analogRead` och `digitalRead` för alla tre sensorer.
2. **Uppdatera internt tillstånd** — edge-detection för knappen.
3. **Bestäm utfall** — en if/else if/else med tre grenar (larm-trigg, stämningsljus, tyst).

4. **Kort paus** — `delay(10)`.

Den fullständiga, rensade och kommenterade sammansatta sketch:en ligger i **Bilaga D** (som delas ut efter hackathonen) — inklusive en `sattFarg(r, g, b)`-hjälpfunktion som gör if/else-grenarna mycket lättare. Behöver ni en referens under hackathonen, fråga kursledaren.

Tröskeln 300 är bara en gissning. Kalibrera själv med Serial Monitor innan hackathonen börjar — värdet hör hemma som en `const int` i toppen så du enkelt kan ändra det.

Vanliga problem och snabblösningar

Larmet flimrar när knappen hålls	Du saknar edge-detection. Återvänd till ”Reagera på flanken” i Modul 3.
Tilten triggar slumpmässigt	Studs. Lägg `delay(50)` efter `digitalRead(tiltPin)`.
RGB-LED blir aldrig riktigt mörk	Glömt att sätta alla tre kanalerna till 0 i ”tyst och mörkt”-grenen. Alla tre kanaler ska vara `analogWrite(pin, 0)`.
Stämningsljuset lyser alltid	Din tröskel är för hög — `ljus < 300` matchar även rumsljus. Sänk till 150 och testa.
Buzzern vägrar bli tyst	En av grenarna sätter `HIGH` men ingen annan återställer till `LOW`. Säkerställ att **varje** gren av if/else skriver både buzzer och LED.
Kompilerar inte — ”variable not in scope”	Variabler deklarerade **inne** i setup() är bara synliga där. Flytta `lastKnappState` och `larmPaslaget` upp till global nivå (utanför setup och loop).

Bygg vidare

Hackathonen är ungefär 2 timmar med paus. Räkna med 60–80 minuter för att få grundlösningen (stegen 1–3 ovan) att köra stabilt. Resterande tid räcker till ett av förslagen nedan för den som vill djupare. Varje förslag är 15–30 minuters extra kod isolerat.

Allt ni behöver är redan i kittet:

- **Dubbelt pip för larm-på/larm-av**. Ett kort pip när larmet aktiveras, två snabba när det stängs av. Förtydligande till användaren vad som hände.
- **Dimning i stället för hopp**. När stämningsljuset tänds — tona upp det över en sekund i stället för att bara slå på. `for`-loop med stegvis ökande `analogWrite`.
- **Hysteres**. Dela tröskeln i två: en ”tänd vid 300, släck vid 400”. Det hindrar LED:en från att blinka av och på när ljuset ligger precis på tröskeln.

– **Förvarning**. När larmet varit på i 30 sekunder, blinka buzzerN i ett mönster som varning om att du glömde stänga av den.
– **Eget morse-meddelande** via `LED_BUILTIN` varje gång tilten triggar.

Vilken som helst av dessa är 15–30 rader extra kod. Prova.

Snabbreferens

Pin-karta	Knapp D9, Tilt D2, LDR A0, Buzzer D12, R/G/B D6/D5/D3.
Kod-skelett	setup: `Serial.begin` + `pinMode` för alla. loop: läs, edge-detect, if/else, styr.
Grundläggande debug	`Serial.print(variabel)` på allt som är mystiskt.
`!larmPaslaget`	Logisk inversion — togglar `true` / `false`.
Full sketch	Bilaga D — delas ut efter hackathonen.

Efter kursen

Fem veckor sedan visste ni inte vad en GND-pinne var. I dag har ni byggt ett system som läser av omvärlden och reagerar på den. **Det är i princip vad embedded-ingenjörer gör på jobbet** — bara i mindre skala och med billigare komponenter.

Nästa steg — om ni vill fortsätta — är inte ett enda steg utan en skog av dem:

– **TinkerCAD Circuits** (`tinkercad.com/circuits`). Autodesks gratis online-simulator för Arduino och breadboard. Ni bygger kretsen i browsern, skriver samma Arduino-kod som i IDE:n, och trycker play — den simulerar hela kretsen inklusive kod, LED:ar, sensorer, serial monitor, allt. Perfekt när ni vill prova en idé på bussen, när kittet står kvar på kontoret, eller när ni vill testa ett koncept innan ni köper en ny komponent. Fungerar i vilken browser som helst.
– **Arduino-projekt online**. Sidor som Hackster, Instructables, Make: Magazine är fulla av recept för allt från väderstationer till MIDI-instrument. Börja enkelt. Kopiera något. Förstå vad du kopierade.
– **Lokala makerspaces**. Det finns ofta ett i närmsta större stad. Gå en kväll. Andras projekt är smittsamt.
– **Fråga er själva: vad i mitt eget hem skulle jag vilja automatisera?** Ett litet problem i verkligheten är mycket bättre motivation än en abstrakt tutorial. ”Lampan vid hallspegeln tänds när jag kommer hem i mörkret.” ”Brevlådan skickar en notis när posten kommit.” ”Kaffebryggaren pausar två minuter efter att jag stängt den av.” Bygg det lilla. Lär dig det stora på vägen.

– **Gå längre in i elektroniken**. De fyra operatorerna från Bilaga A räcker som programmerarpotential för nästan allt Arduino-relaterat. Begränsningen är snart inte koden — den är era idéer om vad som borde göras.

Om ni kommer tillbaka och visar upp något ni byggt: det är kursens högsta beröm. Detta kompendium är er utgångspunkt, inte er slutpunkt.