ADVANCED

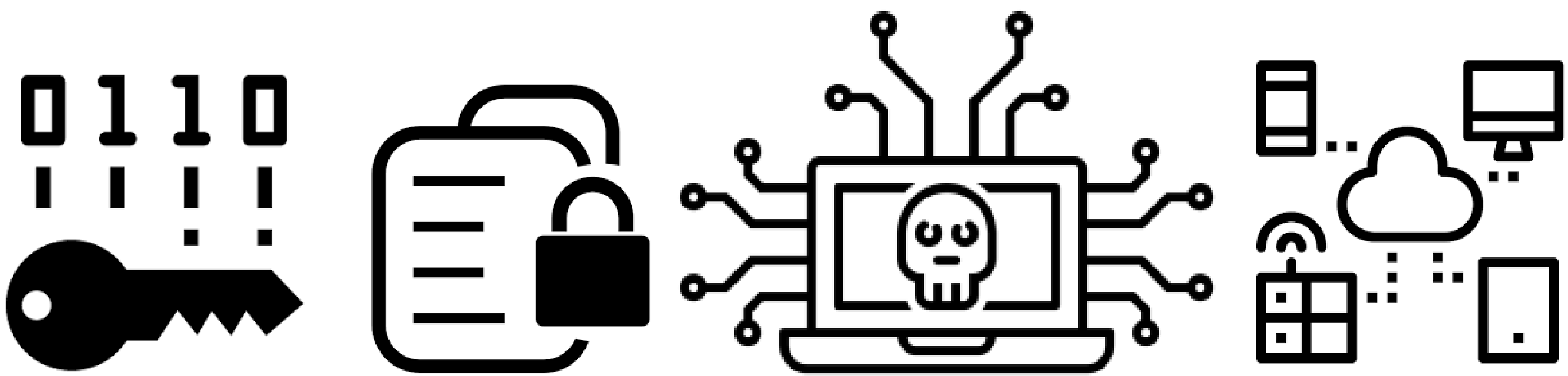
KodeKLIX® Plug-in Required*

Kode KLIX®

Coding Kits For Kids

Cyber Data & Security

Advanced Projects Guide



ADVANCED

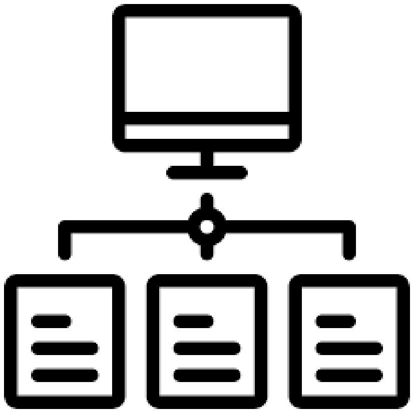
KodeKLIX® Plug-in Required*

Introduction

- This KodeKLIX® project series focuses on the cyber security theme examining technologies and challenges such as:
  - Data transmission and networks
  - Data encryption and decryption
  - Cyber attacks and counter-measures
- You will need access to multiple KodeKLIX® SnapCPUs due the networking aspects of these projects, and ideally work in teams.
- These topics are for Advanced Users

Data Networking

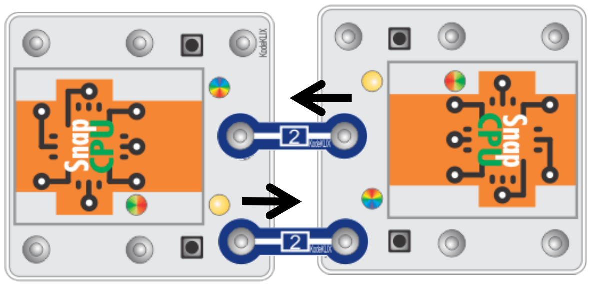
- Data networking includes methods of connecting computing devices to share information
- Network connections can be wired or wireless. Protocols describe the rules through which these devices communicate.



- Some networks are directly connected, whilst some share information openly
  - In shared networks data must be secured

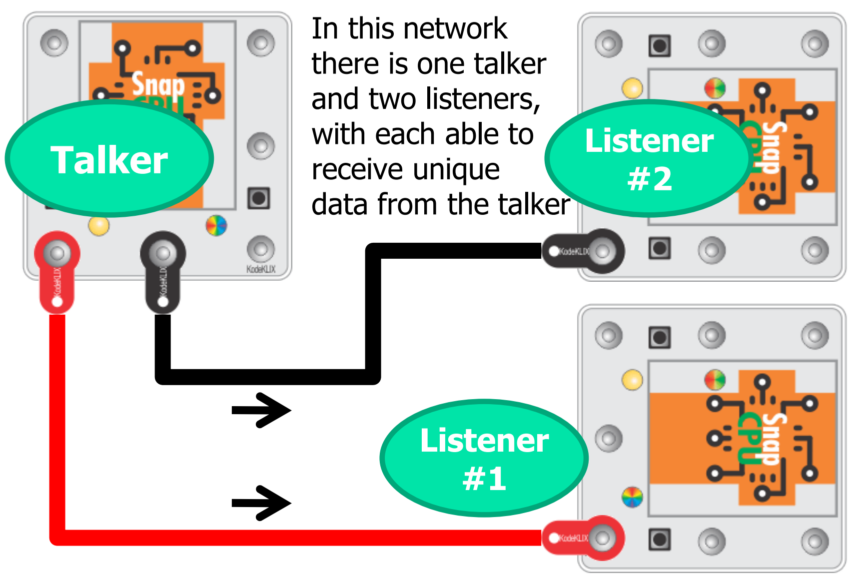
SnapCPU Network

- A network of SnapCPUs can also communicate between each other

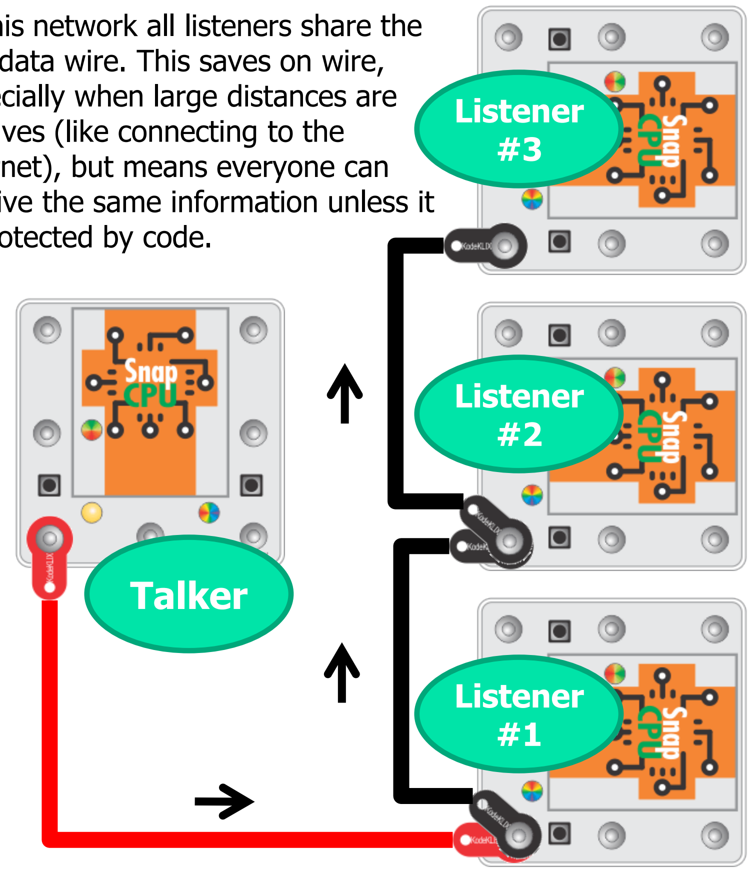


These two SnapCPUs can both send and receive data between each other.

In this network there is one talker and two listeners, with each able to receive unique data from the talker



In this network all listeners share the one data wire. This saves on wire, especially when large distances are involves (like connecting to the internet), but means everyone can receive the same information unless it is protected by code.

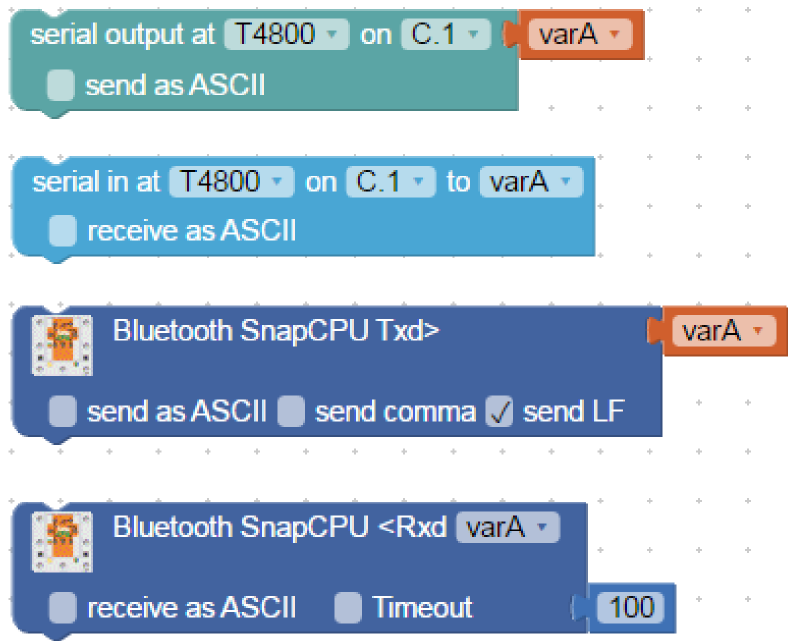


Networking SnapCPUs

- SnapCPUs can communicate with devices like the RFID, DM1 and DM2 displays, even to the laptop for downloads, debug
- The easiest method of communication is using the **serial protocol** known as RS232
- The RS232 protocol allows for one **talker** but multiple **listeners**
- Separate wires are used for talking and listening; RS232 can also be done wirelessly with a special adaptor

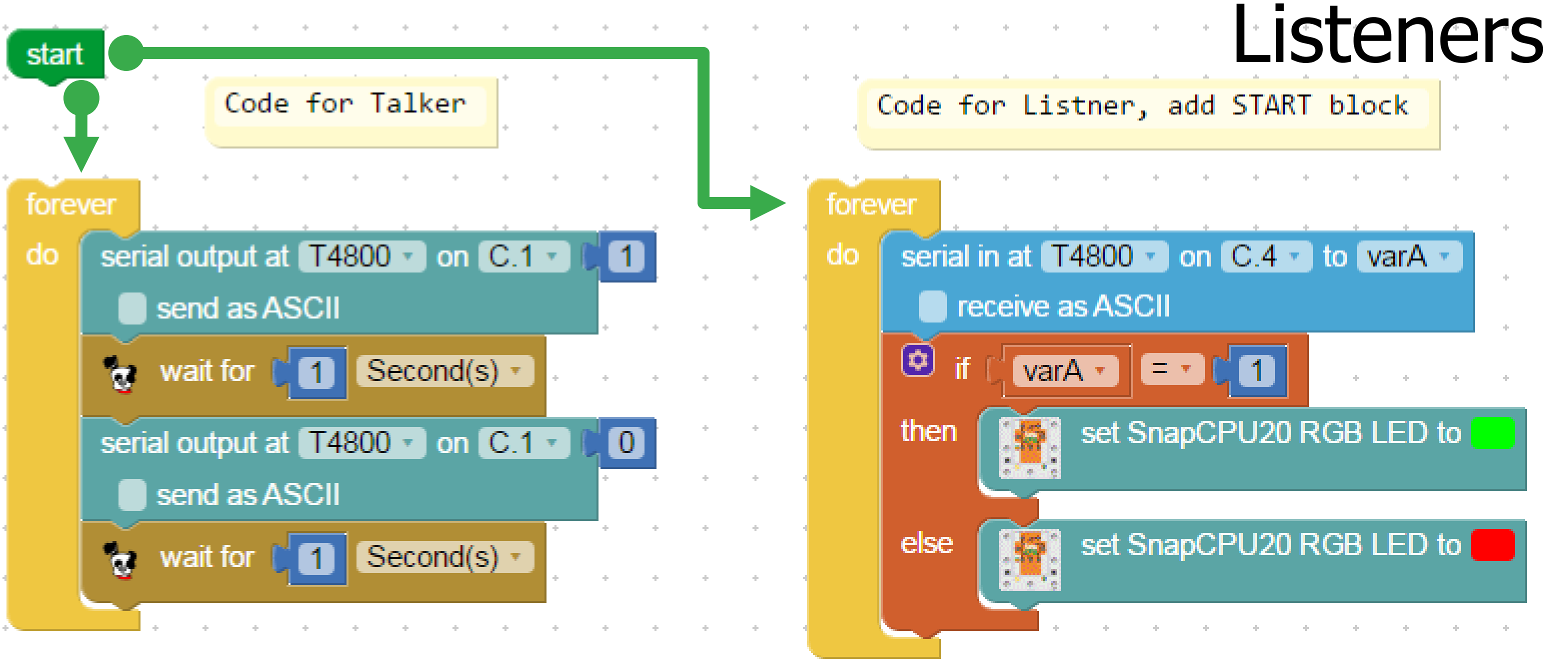
Networking Coding

- For SnapCPUs to communicate by RS232, the coding blocks below are used
- For communication to be effective, the protocol details need to be consistent
  - Serial speed; T4800 or N4800 is a good choice for reliable data transmission
  - ASCII or not; determines format of the data



Understanding Examples

- For the following SnapCPU networking projects to work, code needs to be downloaded to each SnapCPU individual
- Different code is needed for Talkers and Listeners

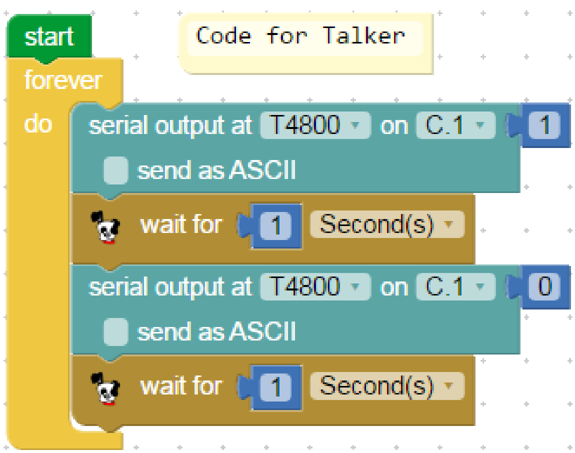


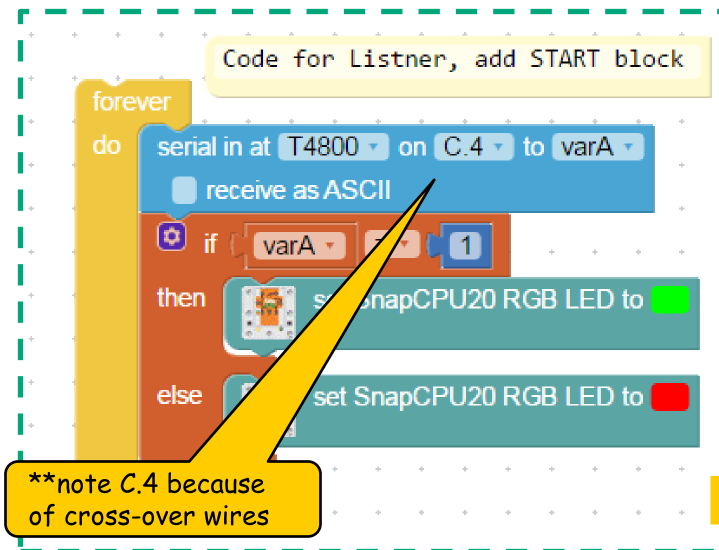
Completed ☐

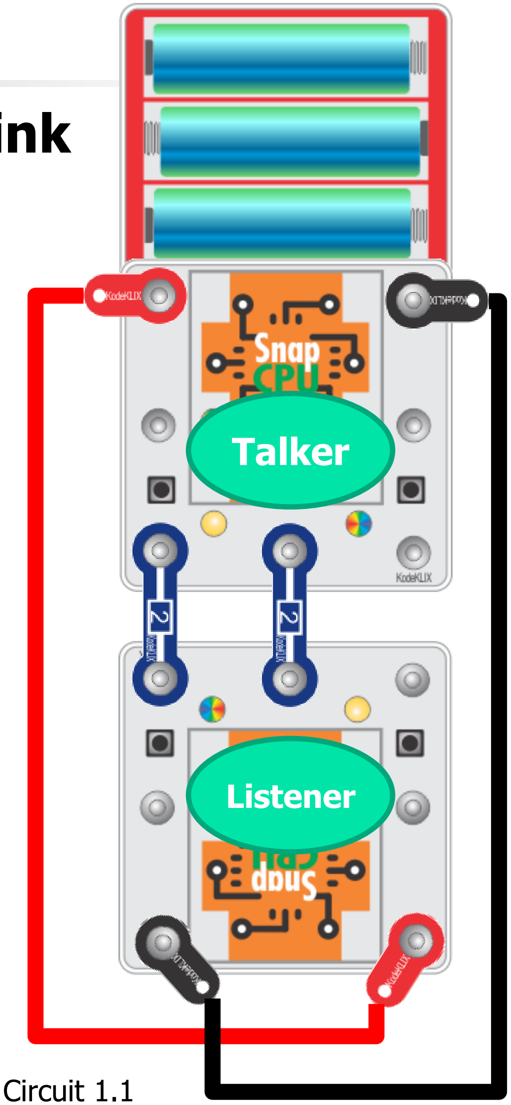
Example Project 1.1

SnapCPU Network Connections – Direct link

- Assemble the snap components as shown in circuit 1.1; Construct the code below and download to talker code to the talker SnapCPU; and listener code to the listener SnapCPU (do this separately)
- The battery pack [70] is for powering the circuit (or you can power with USB)







Circuit 1.1

Always remember to power up all SnapCPUs!

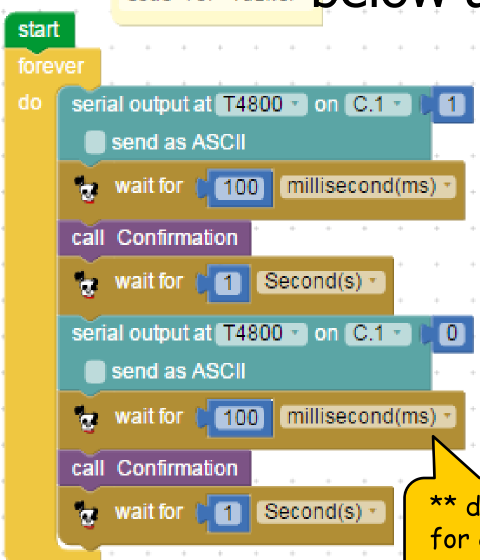
Completed

Example Project 1.2

SnapCPU Network Connections – Feedback

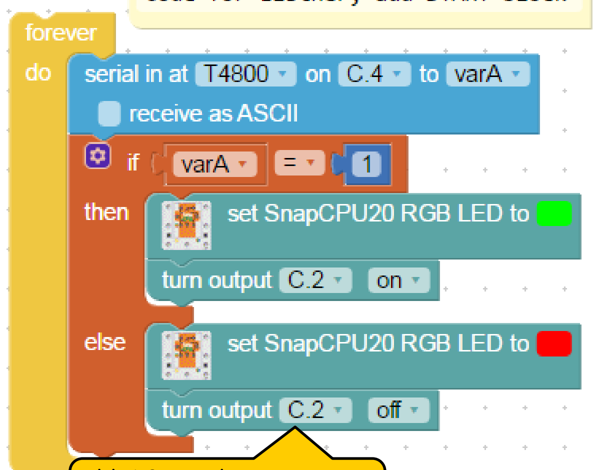
- Assemble the snap components as shown in circuit 1.2; Construct the code below and download to each SnapCPU

Code for Talker

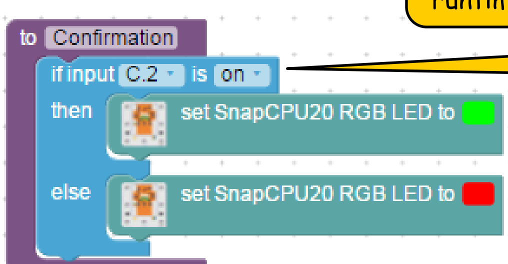


** delay for code runtime

Code for Listner, add START block

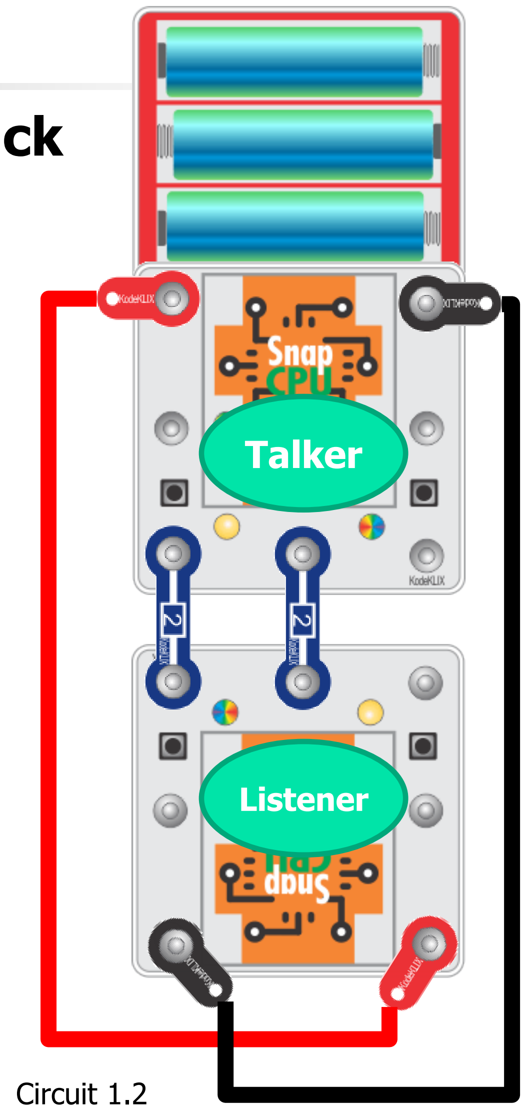


**C.2 used to sent signal back to talker



** Talker checks C.2 for confirmation feedback

Feedback signals vs communications:
In this example a simple yes/no or on/off is used. Sending serial data gets a little more complicated as a listener cannot talk and listen at the same time.



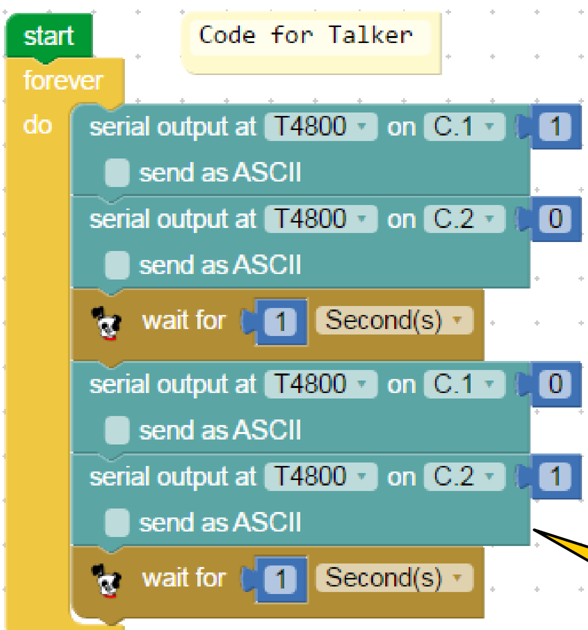
Circuit 1.2

Always remember to power up all SnapCPUs!

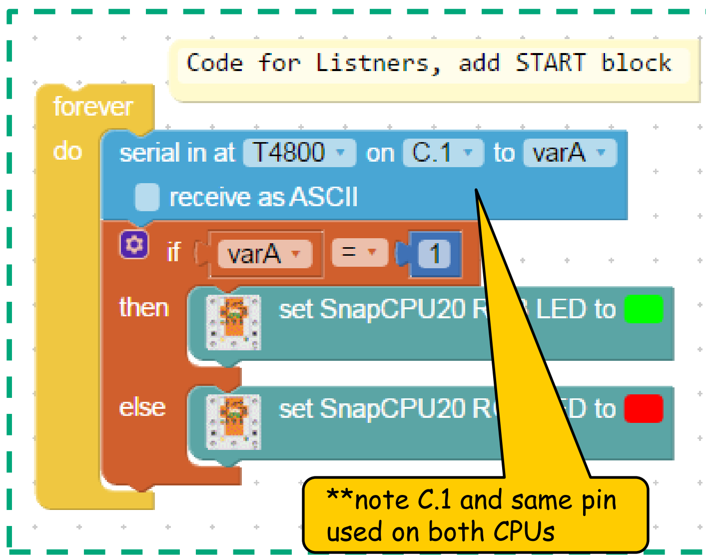
Example Project 1.3

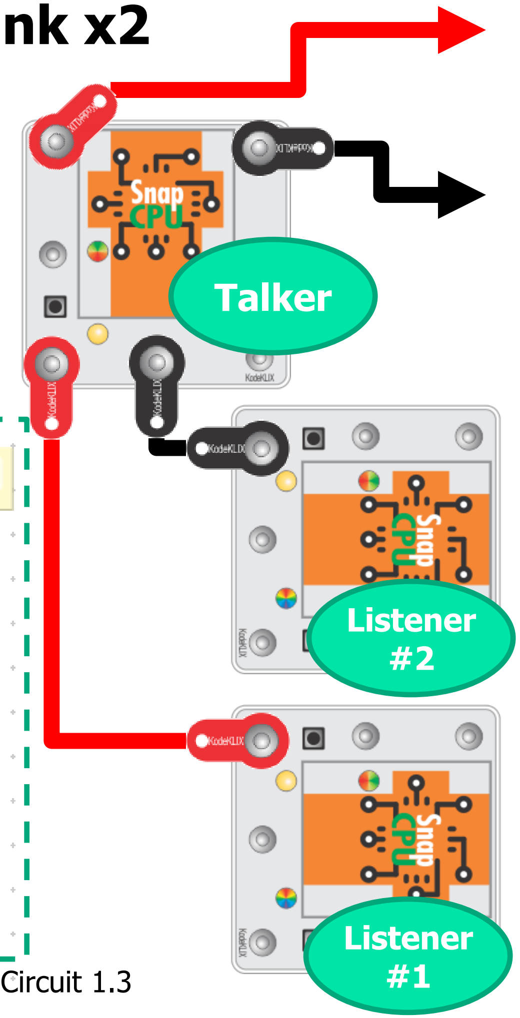
SnapCPU Network Connections – Direct link x2

- Assemble the snap components as shown in circuit 1.3; Do not forget to connect power to all CPUs (not shown, use wires)
- Construct the code below and download to each SnapCPU separately



**different data can be sent to each Listener!





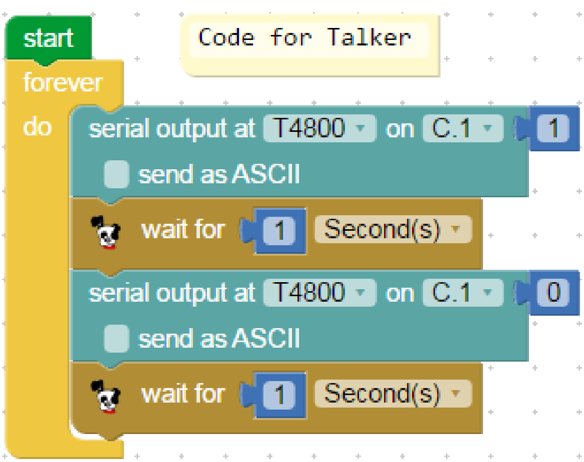
Circuit 1.3

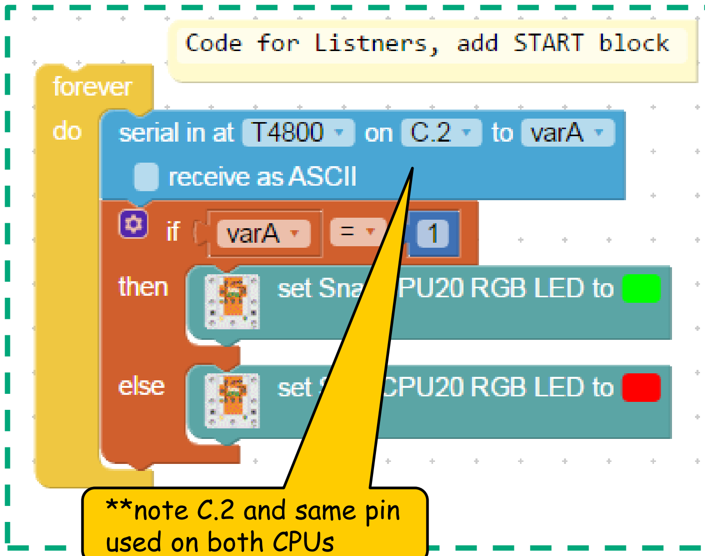
Always remember to power up all SnapCPUs!

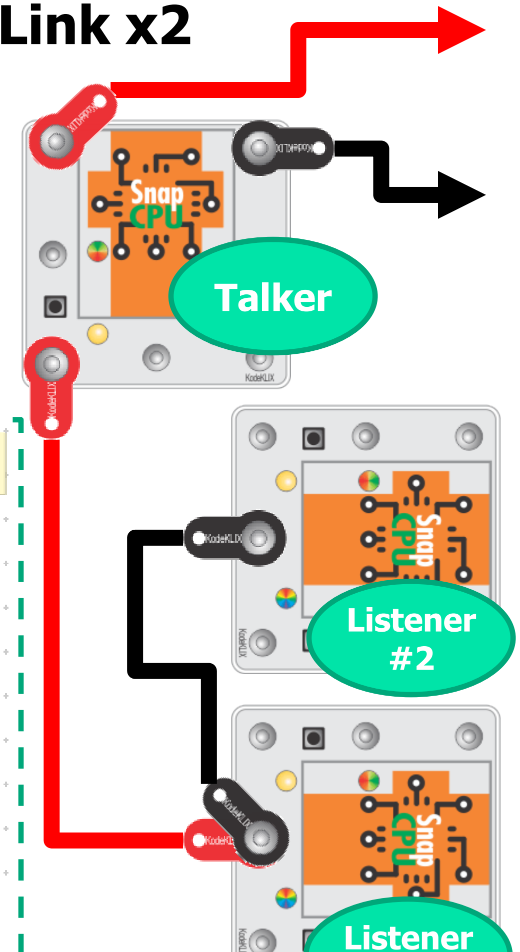
Example Project 1.4

SnapCPU Network Connections – Shared Link x2

- Assemble the snap components as shown in circuit 1.4; Do not forget to connect power to all CPUs (not shown, use wires)
- Construct the code below and download to each SnapCPU separately







Circuit 1.4

Always remember to power up all SnapCPUs!

Network Data Integrity

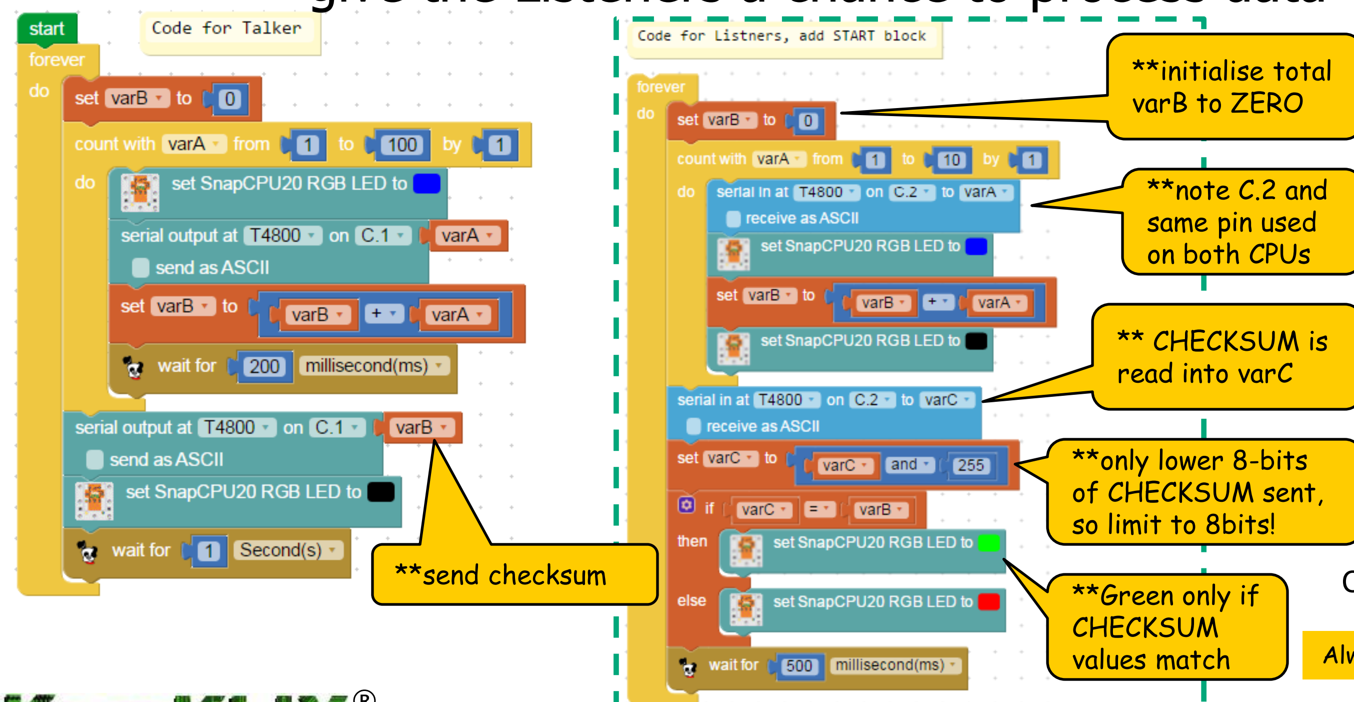
- Network Data during transmission can sometimes become corrupted
  - The ultimate way to check data integrity is to send the information a second time and compare the data between transmissions for errors; this is slow and inefficient
- Two alternate methods are:
  - CHECKSUM – checking what value the data stream 'adds up' to; detecting 1-bit errors
  - CRC – can detect more errors than **checksum** but is a more complex function

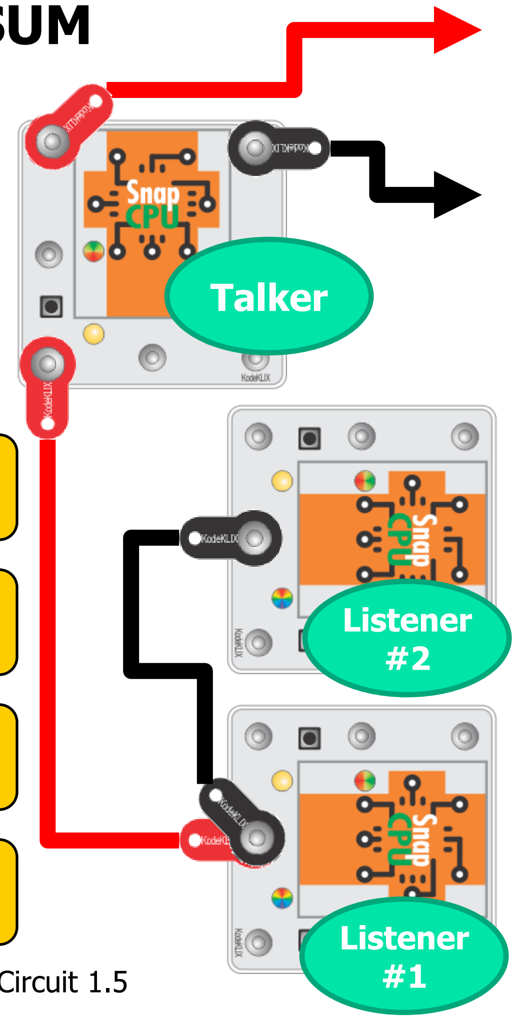
Completed ☐

Example Project 1.5

SnapCPU Network Connections – CHECKSUM

- Circuit 1.5 is the same as circuit 1.4
- Construct the code below and download to each SnapCPU separately
- Review the code; note pauses are used to give the Listeners a chance to process data





Circuit 1.5

Always remember to power up all SnapCPUs!

Network Challenges

CRC Data checking

- CRC, or Cyclic Redundancy Checking, is a fairly complex algorithm
- Rather than try to code the algorithm, use the internet to read up about the different methods used to calculate the error check code and why CRC32 is preferred

Hint: use Google or another search tool to research, or
https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Data Acquisition & Statistics

- Data can be acquired from a number of sources including:
  - People, eg surveys, questionnaires, etc
  - Sensors, eg machine or environmental
- Data when stored becomes valuable if statistically analysed:
  - Averages, maximum, minimum
  - Modelling to predict what comes next
  - Modelling to predict what is not known or understood

Data Acquisition & Statistics

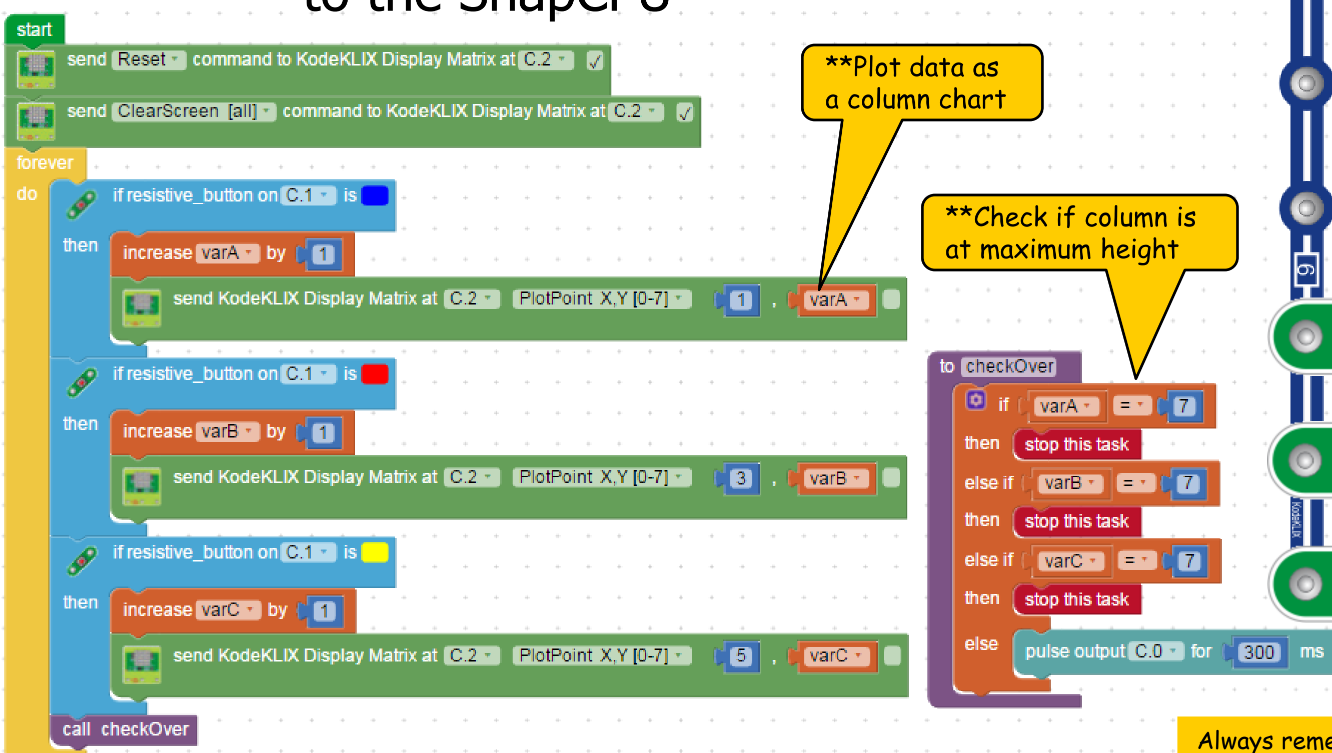
- The KodeKLIX project guides provide insights into a range of data sampling situations and configurations
  - with most of these, the data was sent to the connected computer for analysis
- In this chapter we will extend those project skills to allow the SnapCPU to store and analyse this data realtime, display results and undertake statistics

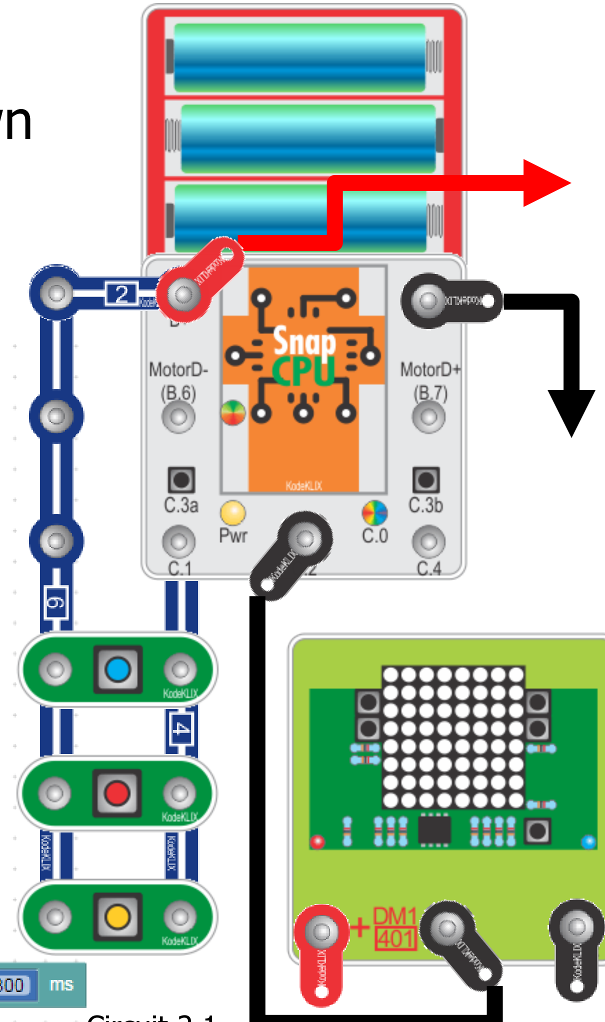
Completed

Example Project 2.1

SnapCPU based Survey Kiosk

- Assemble the snap components as shown in circuit 2.1
- Construct the code below and download to the SnapCPU





Circuit 2.1

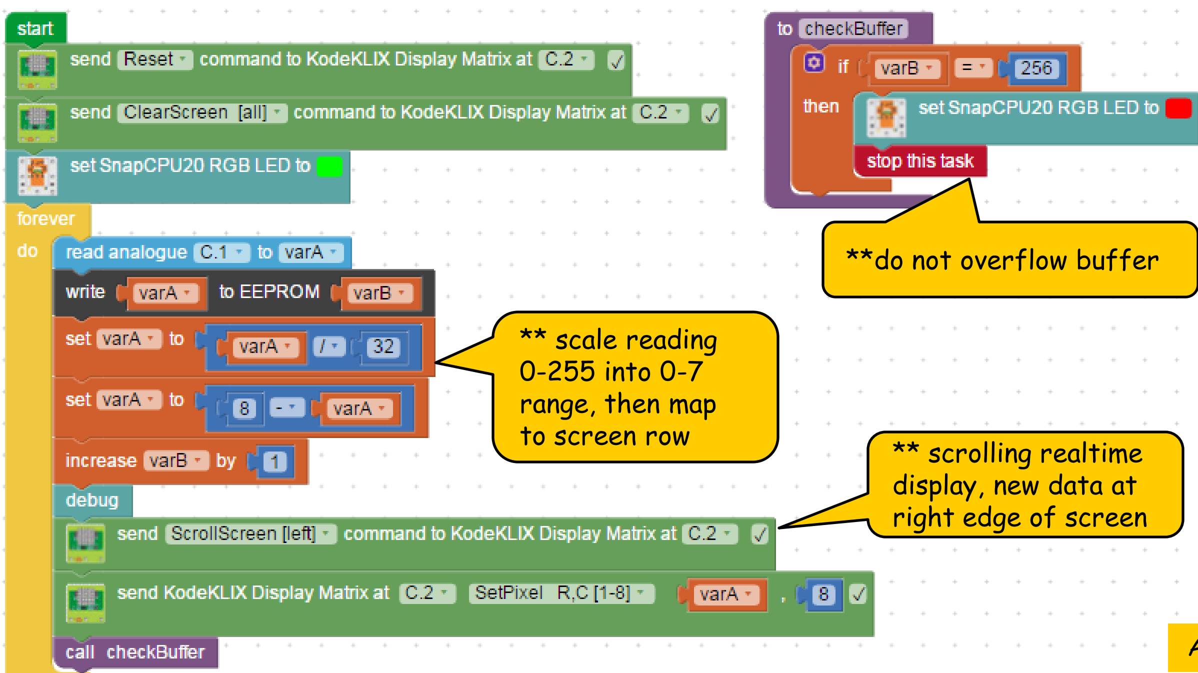
Always remember to power up all SnapCPUs and displays!

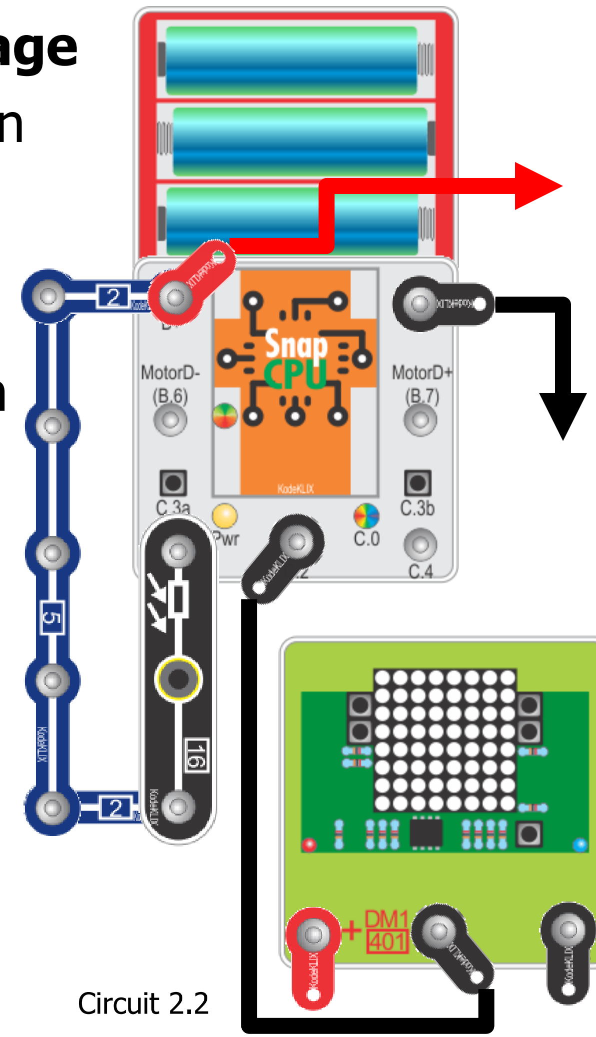
Completed ☐

Example Project 2.2

SnapCPU based Data logger with Storage

- Assemble the snap components as shown in circuit 2.2; Construct the code below and download to the SnapCPU
- Expose the sensor to different levels of light and watch the results on the screen





Circuit 2.2

Always remember to power up all SnapCPUs and displays!

Data Challenges

Multi-Function DATA RECORDER

- Combine circuits 2.1 and 2.2 to produce a tool which uses the three buttons to start, stop and replay ambient brightness data measured by the Light Sensor [16]
  - Upgrade the tool to record light data over a 12hr daytime duration
  - Modify the code to plot temperature using Thermistor [33T] over a 24hour period

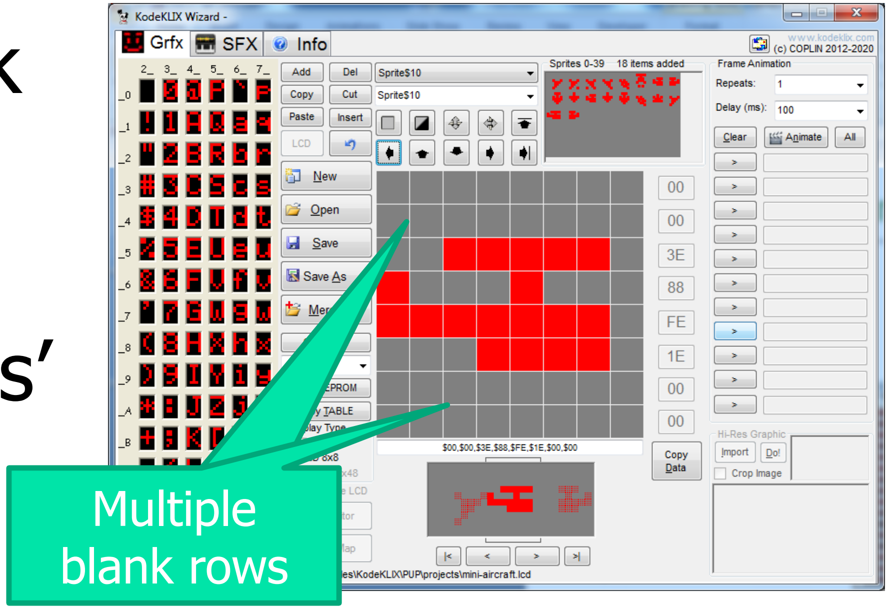
Hint: you only have 256 slots in your data buffer!
Remember: batteries may not last 12hrs, plug in USB!

Data Compression #1

- In today's world data is everywhere; so much data that the internet would grind to a halt if it wasn't for Data Compression
- Consider **streaming** a movie from the internet; the amount of data per picture, the number of pictures per second, the number of seconds in a movie
- In this chapter we will examine a simple movie streaming **protocol** and how compression helps reduce the amount of data that needs to be transmitted

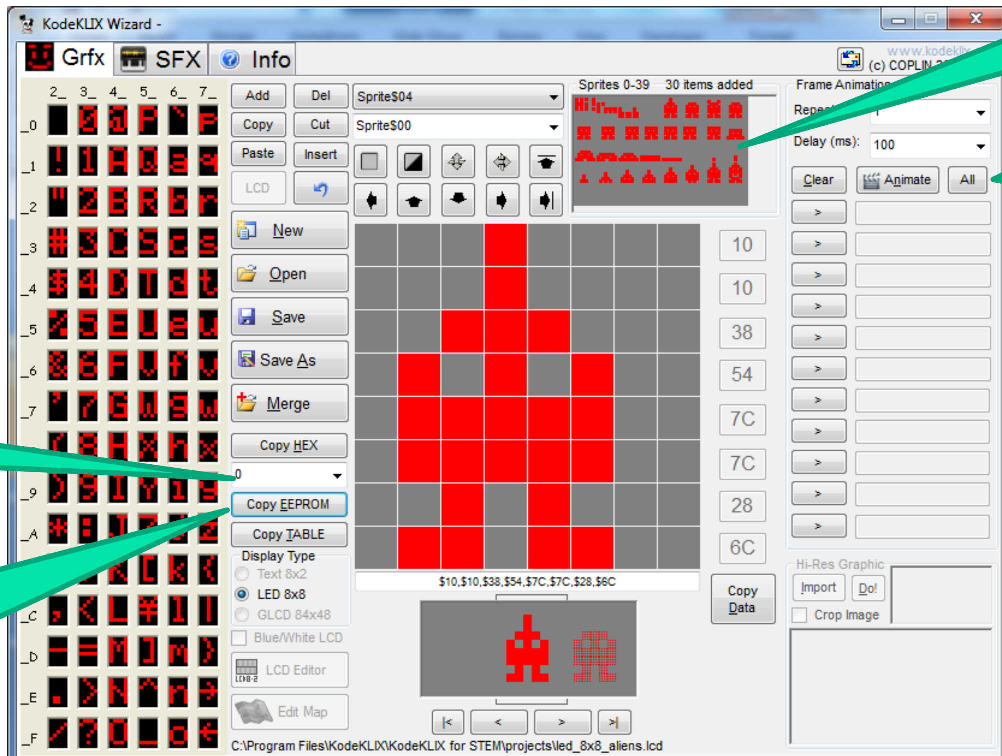
First, create your movie

- Using the KodeKLIX Wizard a movie animation of up to 32 frames (screens) can be created for playback
- For our streaming demonstration, it will be best if your images include some blank rows like show
- Our compression algorithm ‘crunches’ these blank rows



Next, test your movie

- Using the KodeKLIX Wizard the movie can be tested and the data copied for use in your BLOCKLY program



Your movie sequence

This button will animate all of your movie frames

Set this to '0'

Copy all of your movie data

Challenge: a sample movie for the next exercise has been provided, however why don't you try making your own movie? Remember to include lots of blank rows to see the compression algorithm at work

Completed ☐

Example Project 3.1a

SnapCPU uncompressed movie streamer

- Assemble the snap components as shown in circuit 3.1
- Construct the code below and download to the "broadcaster" SnapCPU

```
start
set SnapCPU20 RGB LED to [red]
BASIC  EEPROM 0, ($A9,$A1,$E9,$A8,$A9,$00,$00,$00) 'Sprite$00
       EEPROM 8, ($00,$00,$A0,$80,$9F,$95,$00,$00) '
       EEPROM 16, ($00,$00,$00,$00,$82,$82,$DB,$DB) '
wait for 3 Second(s)
set SnapCPU20 RGB LED to [blue]
count with varA from 0 to 255 by 1
do
    read varB from EEPROM varA
    serial output at N4800 on C.1 varB
        [ ] send as ASCII
    increase varD by 1
    debug
set SnapCPU20 RGB LED to [black]
```

**only some of the data is visible, rest of it on the next page

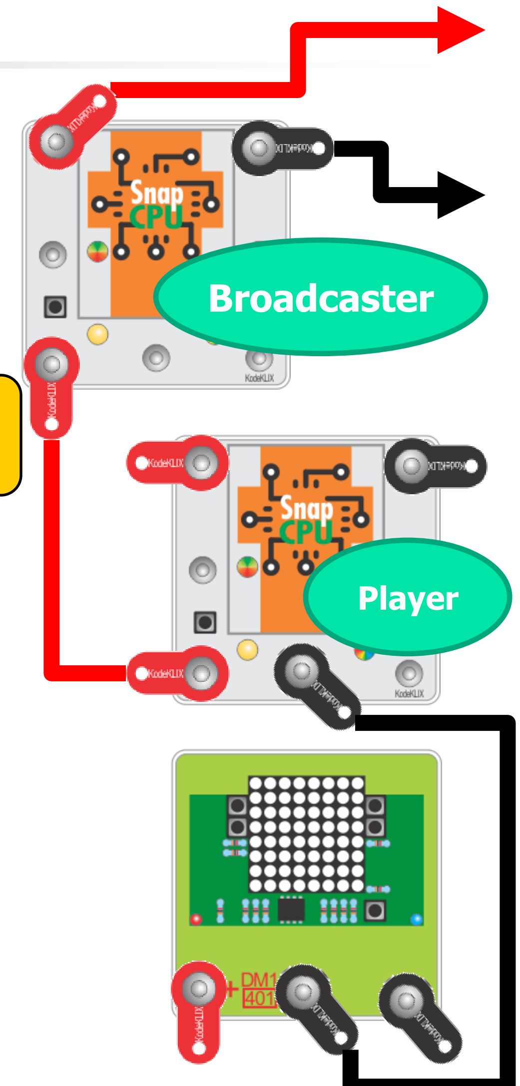
** RBG LED shows when sending data

**each of the 256 data bytes are sent sequentially

**see info next page

**data sent serially byte-by-byte

How many data bytes sent?
varD keeps track of this, use DEBUG to see the result.



Circuit 3.1

Always remember to power up all SnapCPUs and displays!

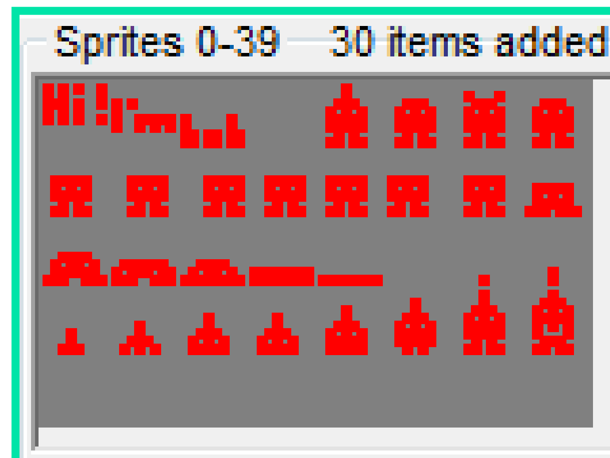
Example Project 3.1a

```
EEPROM 0, ($A9,$A1,$E9,$A8,$A9,$00,$00,$00) 'Sprite$00
EEPROM 8, ($00,$00,$A0,$80,$9F,$95,$00,$00) 'Sprite$01
EEPROM 16, ($00,$00,$00,$00,$82,$82,$DB,$DB) 'Sprite$02
EEPROM 24, ($00,$00,$00,$00,$00,$00,$00,$00) 'Sprite$03
EEPROM 32, ($10,$10,$38,$54,$7C,$7C,$28,$6C) 'Sprite$00
EEPROM 40, ($00,$00,$38,$54,$7C,$7C,$28,$6C) 'Sprite$00
EEPROM 48, ($00,$44,$38,$54,$7C,$7C,$28,$6C) 'Sprite$00
EEPROM 56, ($00,$00,$38,$54,$7C,$7C,$28,$6C) 'Sprite$00
EEPROM 64, ($00,$00,$00,$7C,$54,$7C,$28,$6C) 'Sprite$00
EEPROM 72, ($00,$00,$00,$3E,$2A,$3E,$14,$36) 'Sprite$00
EEPROM 80, ($00,$00,$00,$1F,$15,$1F,$0A,$1B) 'Sprite$00
EEPROM 88, ($00,$00,$00,$3E,$2A,$3E,$14,$36) 'Sprite$00
EEPROM 96, ($00,$00,$00,$7C,$54,$7C,$28,$6C) 'Sprite$00
EEPROM 104, ($00,$00,$00,$F8,$A8,$F8,$50,$D8) 'Sprite$00
EEPROM 112, ($00,$00,$00,$7C,$54,$7C,$28,$6C) 'Sprite$00
EEPROM 120, ($00,$00,$00,$00,$7C,$54,$7C,$EE) 'Sprite$00
EEPROM 128, ($00,$00,$00,$00,$3C,$5A,$7E,$E7) 'Sprite$00
EEPROM 136, ($00,$00,$00,$00,$00,$7E,$BB,$E7) 'Sprite$00
EEPROM 144, ($00,$00,$00,$00,$00,$3C,$5A,$FF) 'Sprite$00
EEPROM 152, ($00,$00,$00,$00,$00,$00,$FF,$FF) 'Sprite$00
EEPROM 160, ($00,$00,$00,$00,$00,$00,$00,$FF) 'Sprite$00
EEPROM 168, ($00,$00,$00,$00,$00,$00,$00,$00) 'Sprite$15
EEPROM 176, ($00,$00,$00,$00,$00,$00,$00,$10) 'Sprite$00
EEPROM 184, ($00,$00,$00,$00,$00,$00,$10,$10) 'Sprite$00
EEPROM 192, ($00,$00,$00,$00,$00,$10,$10,$38) 'Sprite$00
EEPROM 200, ($00,$00,$00,$00,$10,$10,$38,$54) 'Sprite$00
EEPROM 208, ($00,$00,$00,$10,$10,$38,$54,$7C) 'Sprite$00
EEPROM 216, ($00,$00,$00,$10,$10,$38,$54,$7C) 'Sprite$00
EEPROM 224, ($00,$00,$10,$10,$38,$54,$7C,$7C) 'Sprite$00
EEPROM 232, ($00,$10,$10,$38,$54,$7C,$7C,$28) 'Sprite$00
EEPROM 240, ($10,$10,$38,$54,$7C,$7C,$28,$6C) 'Sprite$00
EEPROM 248, ($10,$38,$54,$7C,$54,$44,$38,$6C) 'Sprite$00
```

Storing Lots of Data

The SnapCPU has different ways it can store data. Variables are one way.

When it comes to large amounts of data, this needs to be stored in special memory call EEPROM (can be read and written) or TABLE (read only). This data is stored sequentially byte-by-bye and is accessed using an 'index'.

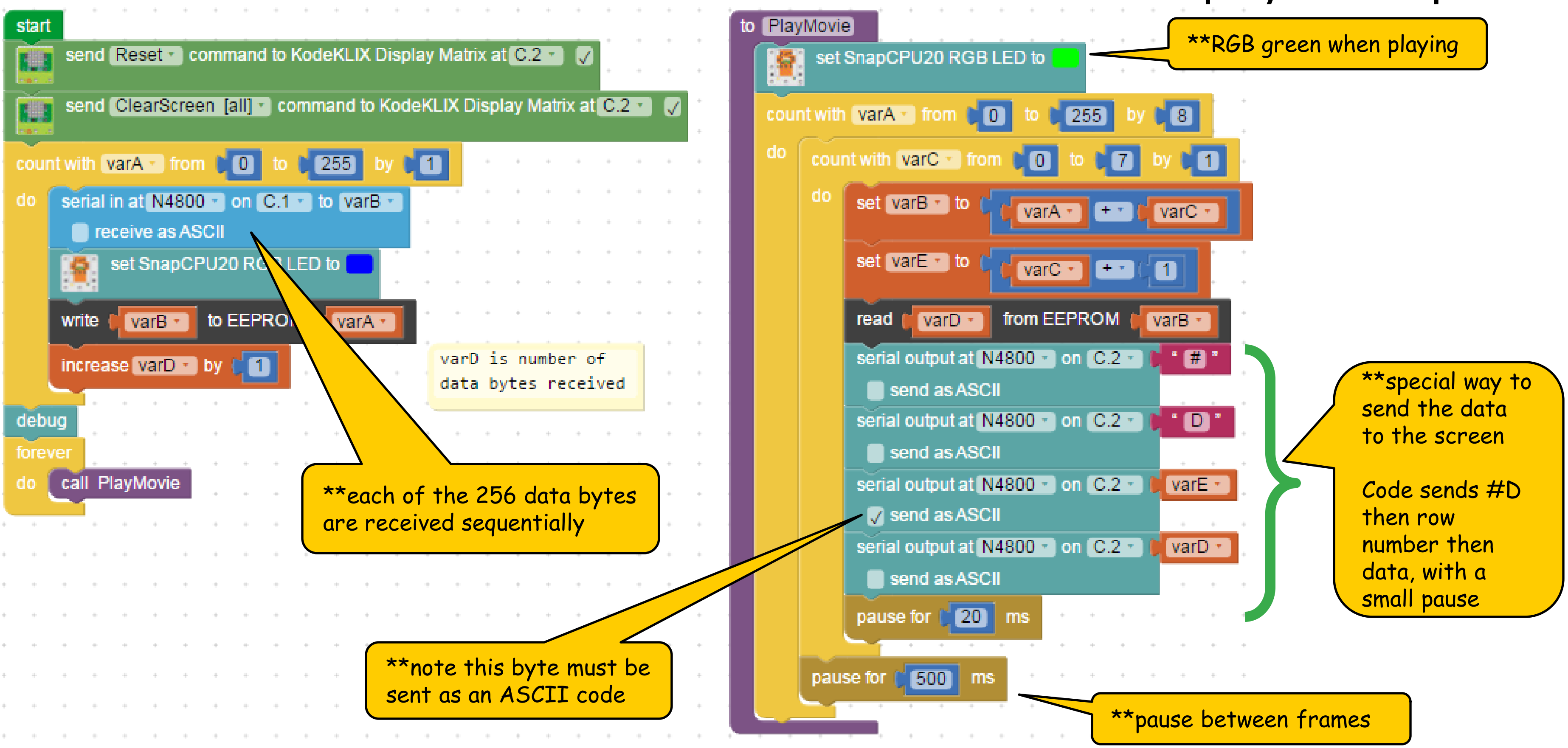


Completed ☐

Example Project 3.1b

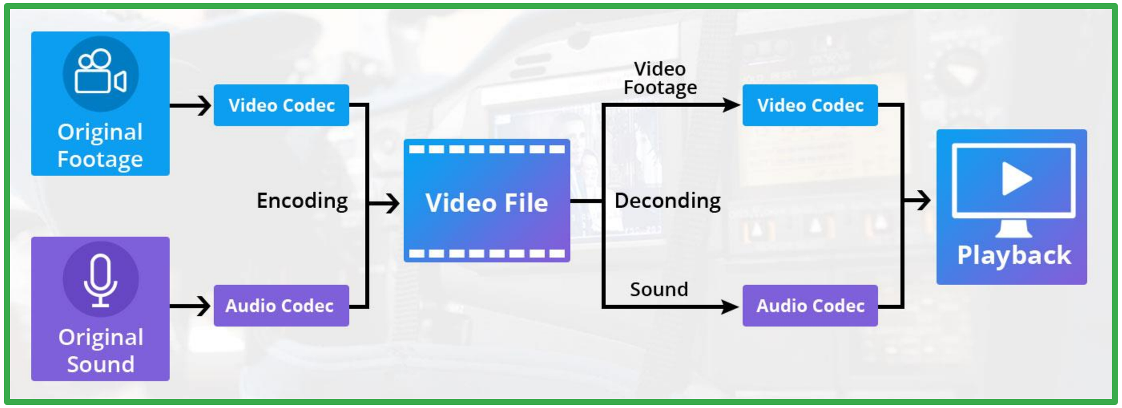
SnapCPU uncompressed Movie Viewer

- Assemble the snap components as shown in circuit 3.1
- Construct the code below and download to the "player" SnapCPU



Data Compression #2

- Our movie streaming data compression **algorithm** is compacts the number of zeros transmitted using this logic:
  - If data is not zero, send it
  - If data is zero, then
    - Send zero followed by
    - Number of zeros in a row



Algorithm and compression ratio:
If only one zero in series is detect, two data bytes are sent, but if three or more zeros in series are detect only two bytes thus reducing the size of the data stream.

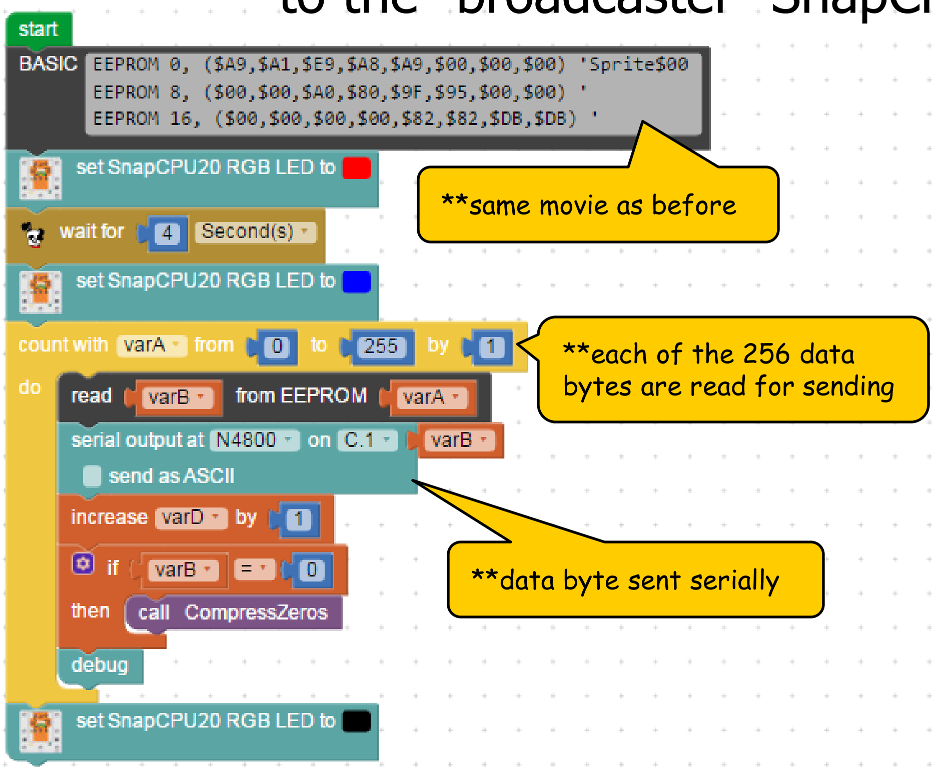
Completed ☐

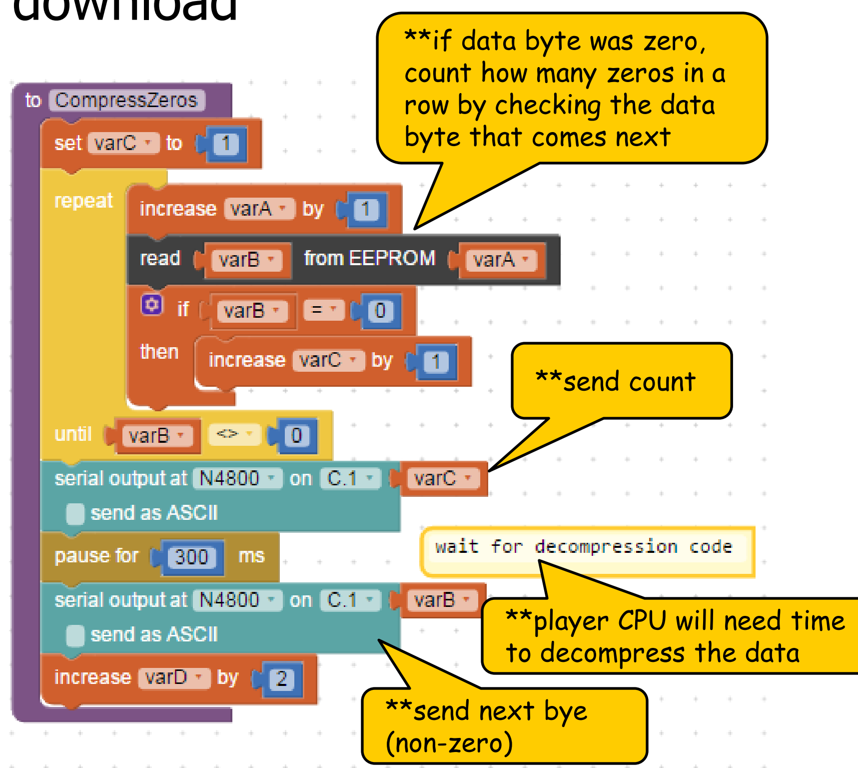
Example Project 3.2a

SnapCPU compressed movie streamer

- Assemble the snap components as shown in project circuit 3.1, previously
- Construct the code below and download to the “broadcaster” SnapCPU

How many data bytes sent?
varD keeps track of this, use DEBUG to see the result. If there are a lot of sequential zeros, varD will be much less than 256.





Data De-Compression

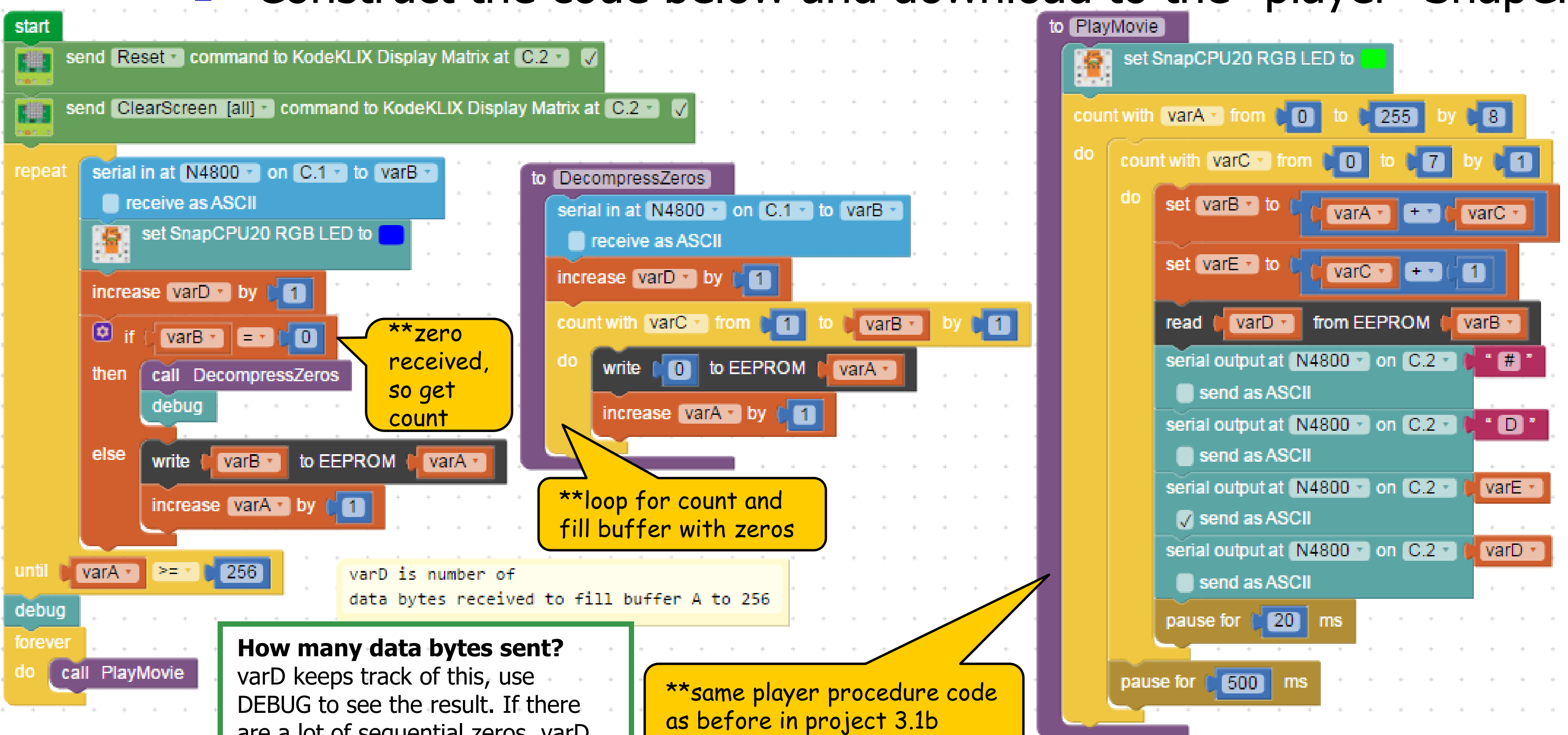
- Our movie player code needs to know how to **de-compress** the data before displaying
- De-compression is the opposite of compression, or unpacking of data
- The logic for the algorithm is like this:
  - If a non-zero is received, save it
  - If a zero is received
    - Read the count data that follows
    - Save count number of zeros in a row
  - Loop back to read next item or stop

Completed

Example Project 3.2b

SnapCPU compressed Movie Viewer

- Assemble the snap components as shown in circuit 3.1
- Construct the code below and download to the “player” SnapCPU



How many data bytes sent?
varD keeps track of this, use DEBUG to see the result. If there are a lot of sequential zeros, varD will be much less than 256.

More Data Challenges

Multi-CODEC Movie Player

- For the circuit in 3.1, update the code to automatically detect what format the movie is (**uncompressed** or **compressed**) and run the correct code procedure

Hint: Codec is a fancy name for movie compression format. Here we only use our custom Codec.

- For your movie, work out what is the next most popular sequential data byte and make a **codec** that compresses two values

Hint: check the EEPROM data for frequency stats!

Data Encryption

- Data encryption is a means of translating data into another form, or code
- This make the data secure so only people with access to a secret key (formally called a decryption key) or password can read it
- There are lots of encryption standards, often referred to by the number of bits, eg 8bit, 64bit, 128bit, etc
  - The more bits, the more secure the code is

Completed ☐

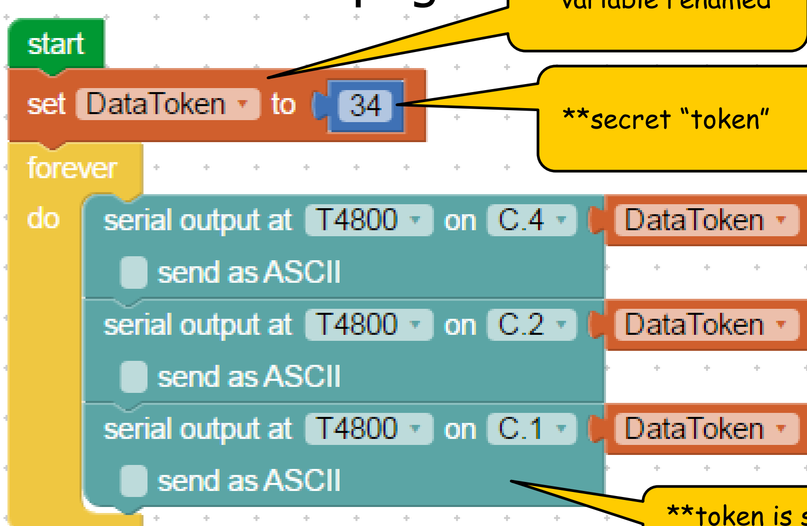
Example Project 4.1a

Transmitting a Data Token (direct connect)

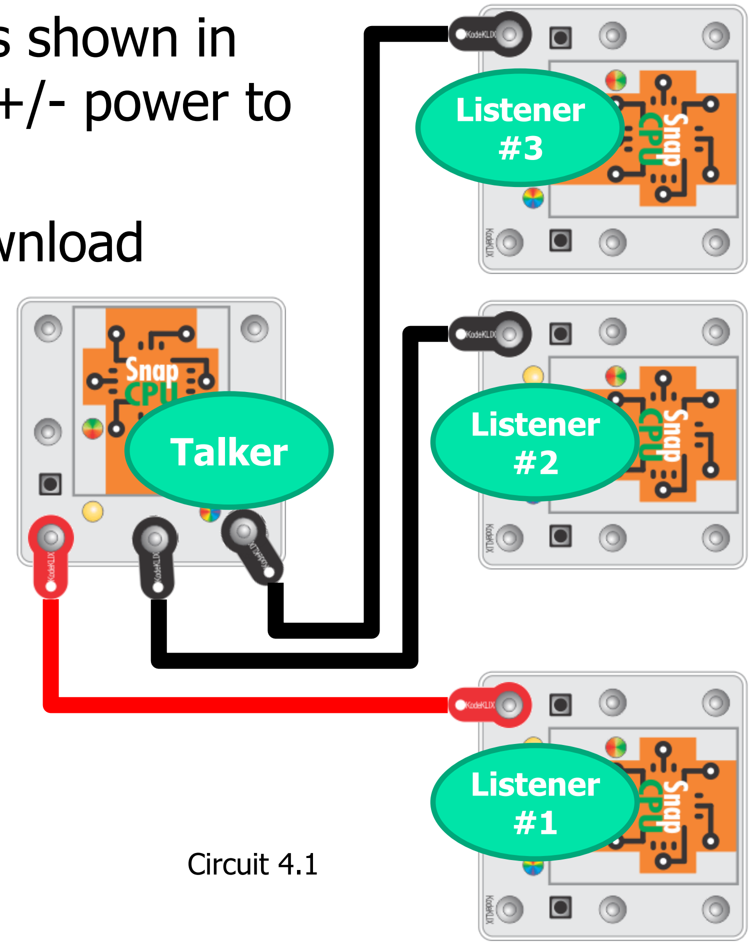
- Assemble the snap components as shown in circuit 4.1; remember to connect +/- power to each SnapCPU
- Construct the code below and download to the "Talker" SnapCPU
- Code for the "Listeners" is on the next page

**variable renamed

**secret "token"



**token is sent by separate wires, so secured by "direct connection" approach



Circuit 4.1

Always remember to power up all SnapCPUs!

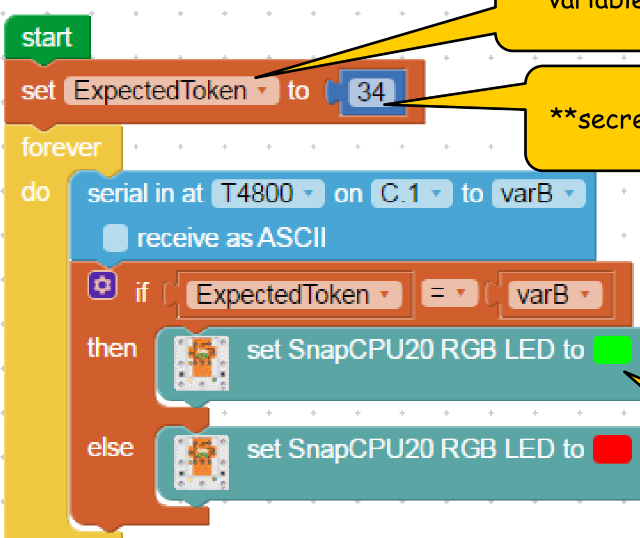
Completed

Example Project 4.1b

Receiving a Data Token (direct connect)

- Construct the code below and download to each of the "Listeners", one at a time

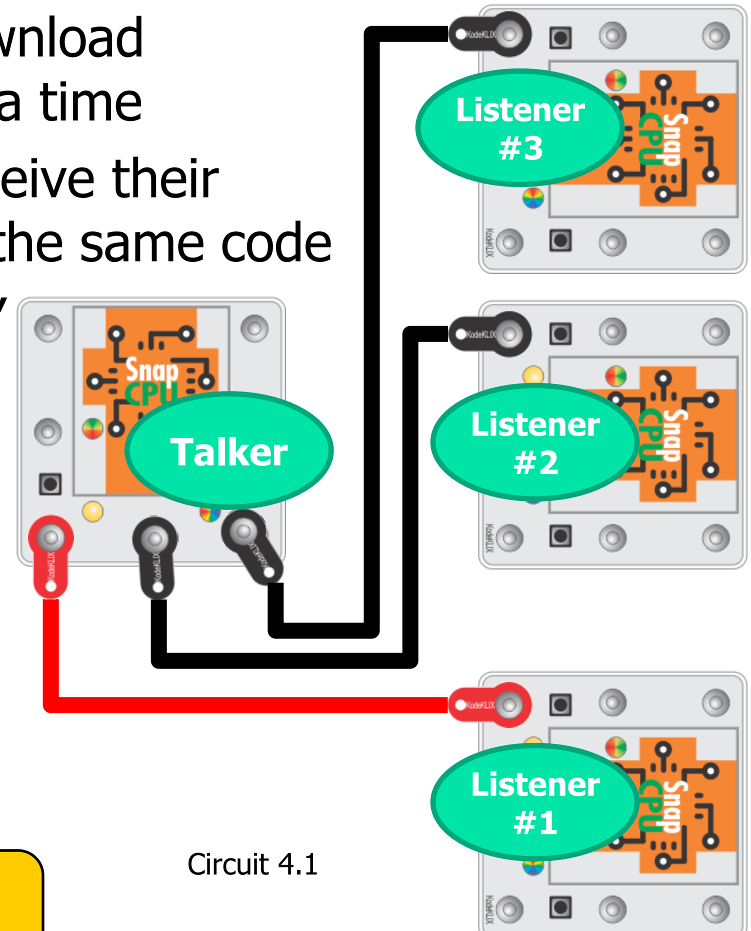
  Note: that since all "Listeners" receive their data from connection C.1, all get the same code
- When powered up, all 3 "Listener" LEDs should glow green



**variable renamed

**secret "token"

**green of received data is the same as the expected token



Circuit 4.1

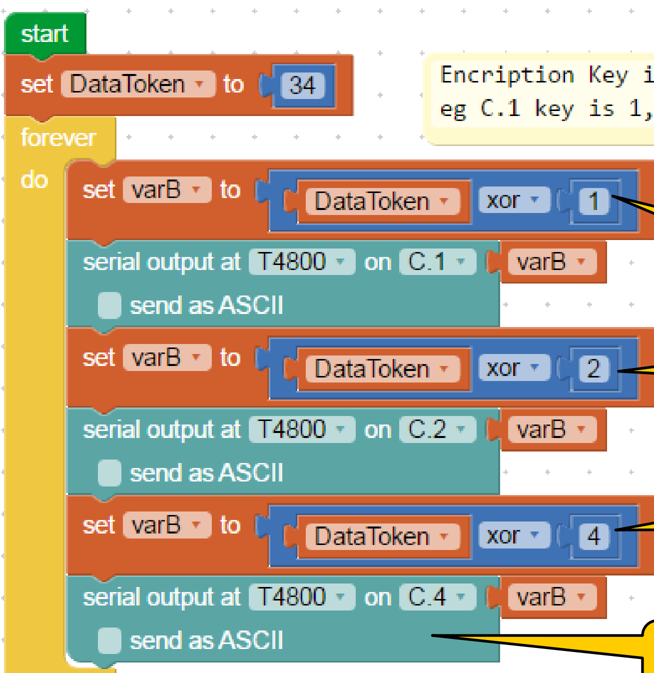
Always remember to power up all SnapCPUs!

Completed ☐

Example Project 4.2a

Transmitting a Data Token (8bit XOR encryption)

- Assemble the snap components as shown in circuit 4.2; remember to connect +/- power to each SnapCPU
- Construct the code below and download to the "Talker" SnapCPU



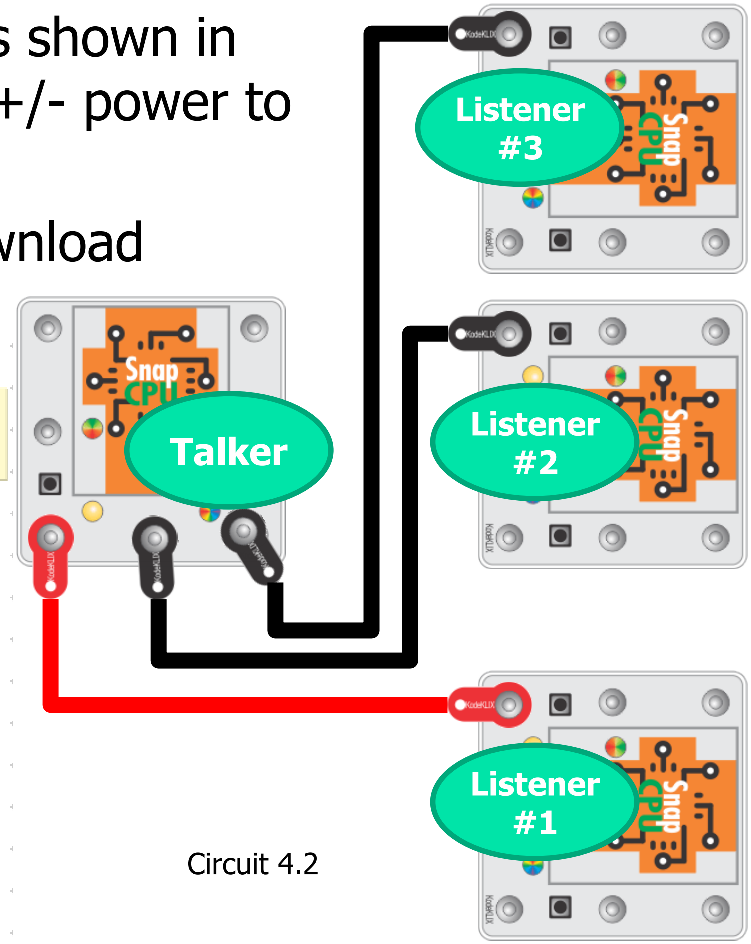
Encription Key is simply the connection number
eg C.1 key is 1, C.2 is 2, C.4 is 4

**unique encryption key for Listener #1

**unique encryption key for Listener #2

**unique encryption key for Listener #4

**token is sent by separate wires, so secured by "direct connection" approach



Circuit 4.2

Always remember to power up all SnapCPUs!

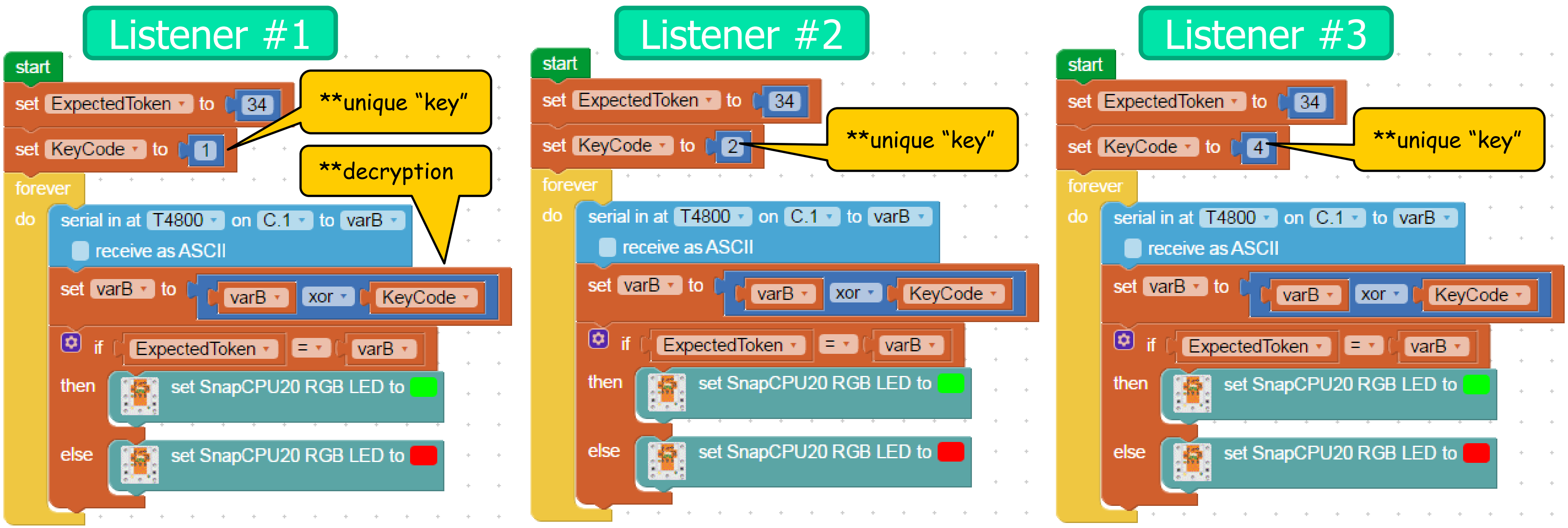
Completed ☐

Example Project 4.2b

Receiving a Data Token (8bit XOR decryption)

- Construct the code below and download to each of the “Listeners”, one at a time

  Note: since each “Listener” has a unique decryption key, the each get special code
- When powered up, all 3 “Listener” LEDs should glow green



More about XOR…

- XOR is a very useful Boolean function that was discussed in KodeKLIX Project Guide Book 2, chapter 3
  - 8-bit XOR Encryption example

    Data=01100011 (99)
    Key=01010101 (85)

    Encoded=0110110 (54)
  - 8-bit XOR Decryption example

    Data=0110110 (54)
    Key=01010101 (85)

    Decoded=01100011 (99)

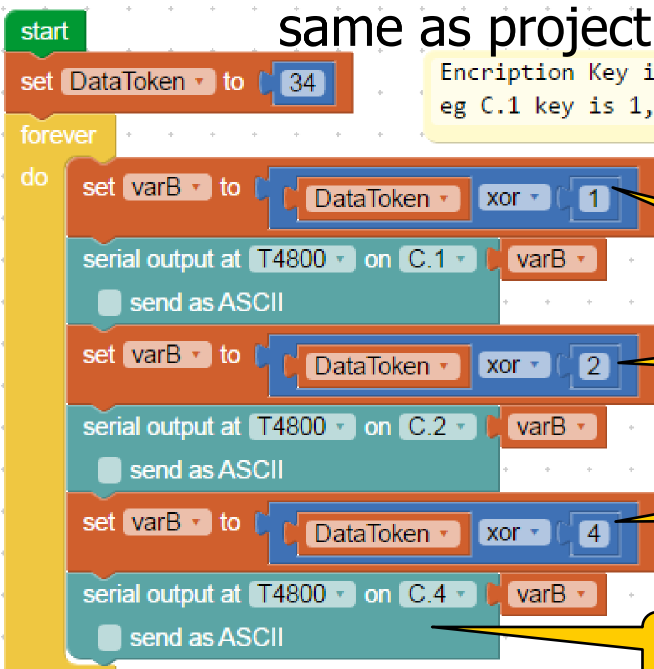
Data Protection

- With an ever-increasing need to protect information, systems are improving how data is protected
  - More and more, secondary and even tertiary levels of protection is used in addition to encryption including MFA (multi-factor authentication)
  - One method is hardware keys in addition to software keys; this may be in the form of a key switch, USB dongle or RFID card

Example Project 4.3a

Transmitting a Data Token (8bit XOR encryption + MFA)

- Assemble the snap components as shown in circuit 4.3; remember to connect +/- power to each SnapCPU. The switches [15] are the hardware key in this project
- The "Talker" SnapCPU code is the same as project 4.2a



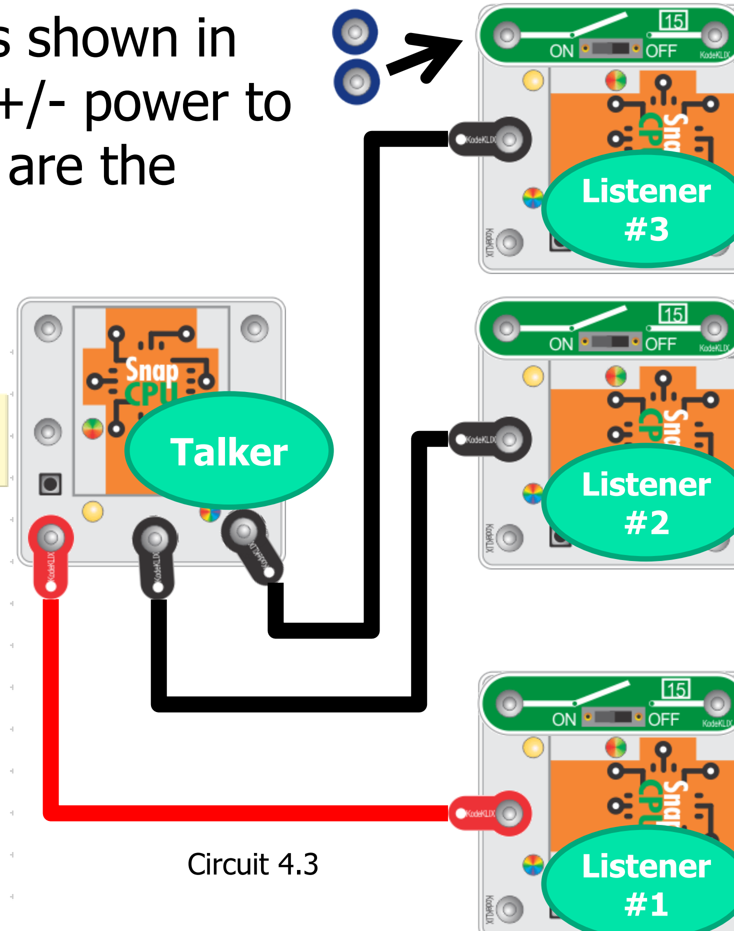

```
Encription Key is simply the connection number
eg C.1 key is 1, C.2 key is 2, C.4 key is 4
```

**unique encryption key for Listener #1

**unique encryption key for Listener #2

**unique encryption key for Listener #4

**token is sent by separate wires, so secured by "direct connection" approach



Circuit 4.3

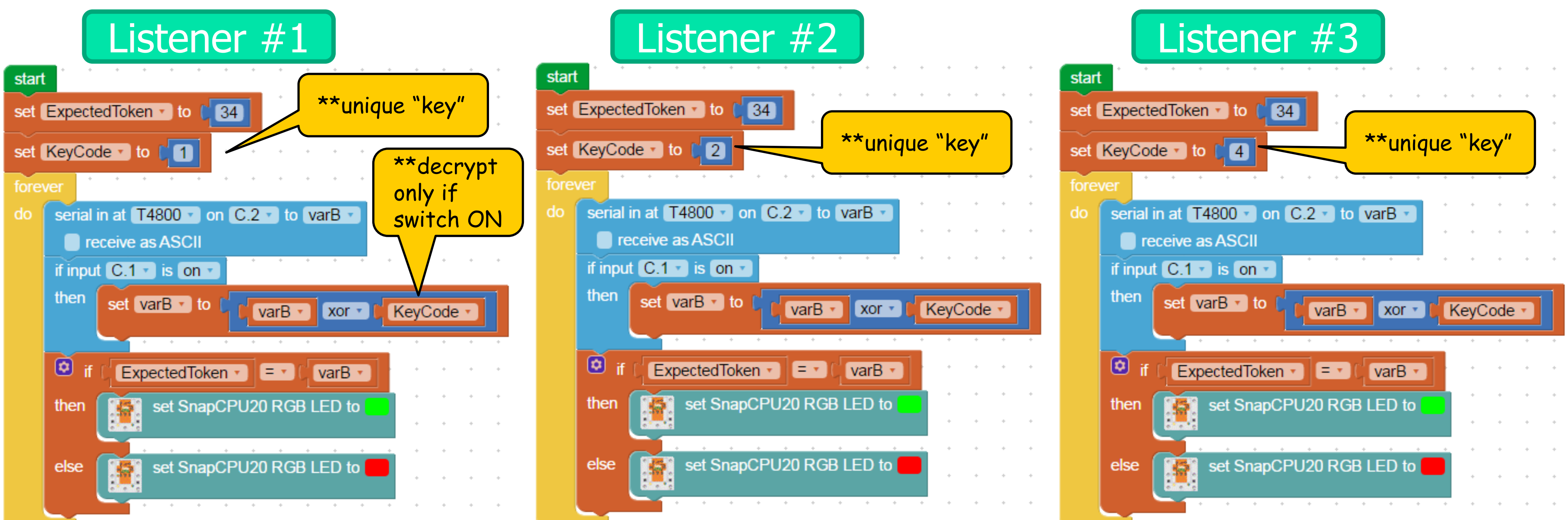
Always remember to power up all SnapCPUs!

Completed ☐

Example Project 4.3b

Receiving a Data Token (8bit XOR decryption+MFA)

- Construct the code below and download to each of the "Listeners", one at a time
- When powered up, "Listener" LEDs glow green only when hardware switch [15] is also in the "ON" position
- The switch is the second key; a hardware key

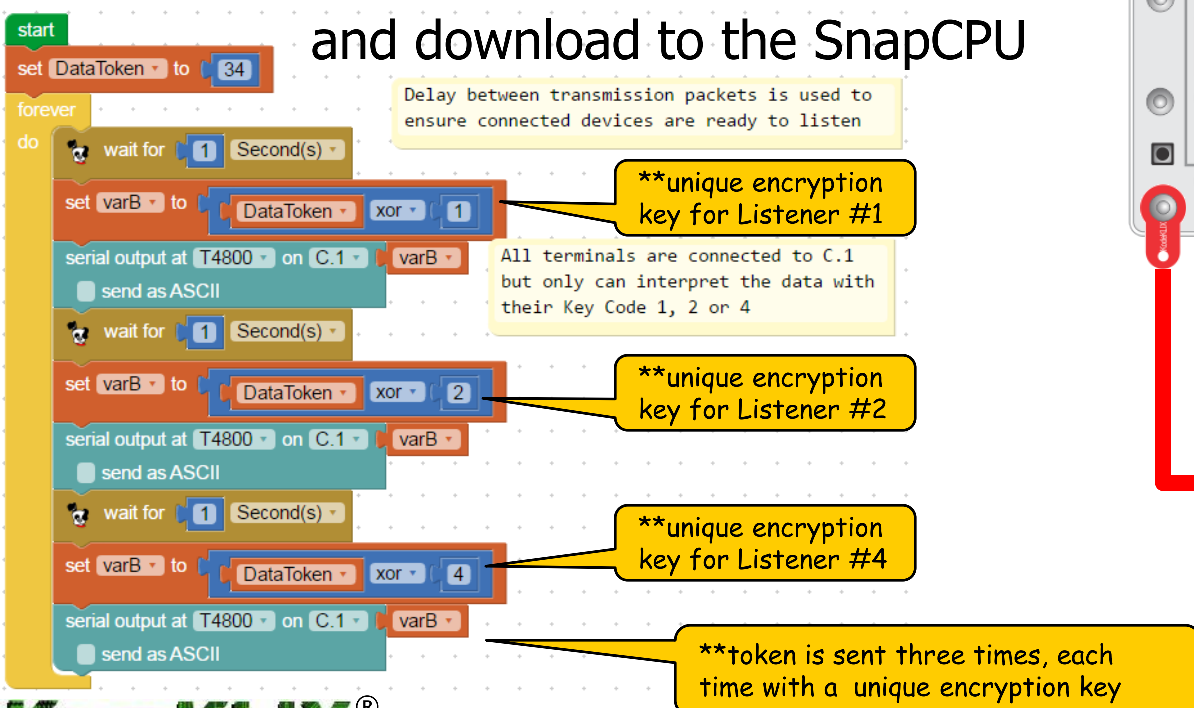


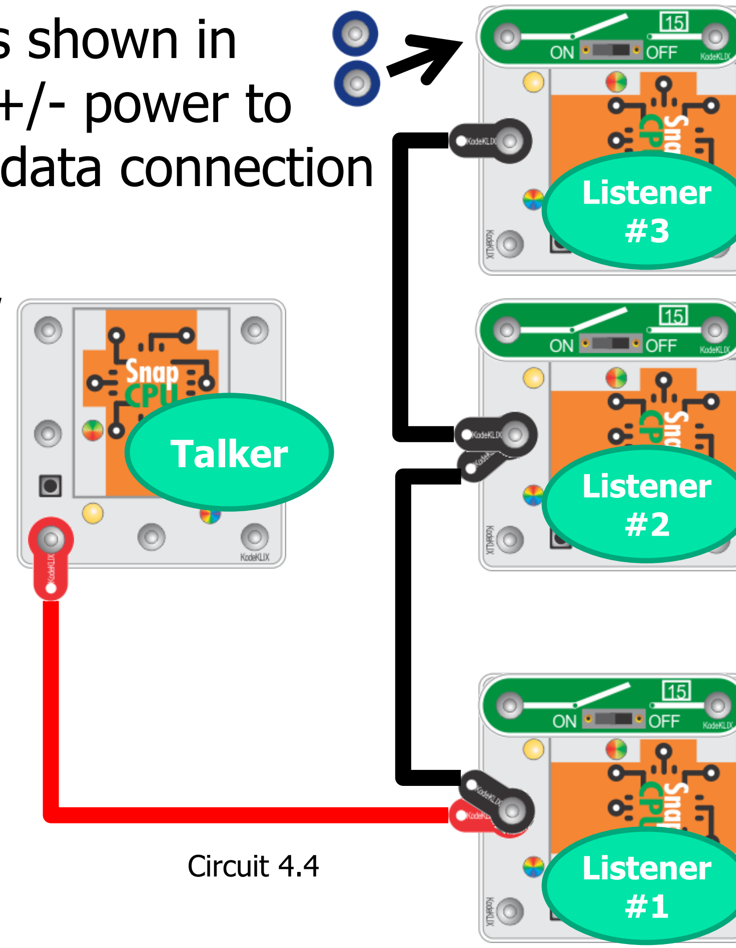
Completed ☐

Example Project 4.4a

Transmitting a Data Token (8bit packet encryption + MFA)

- Assemble the snap components as shown in circuit 4.4; remember to connect +/- power to each SnapCPU. This time the one data connection is shared to all the "Listeners"
- Construct the "Talker" code below and download to the SnapCPU





Circuit 4.4

Always remember to power up all SnapCPUs!

Completed ☐

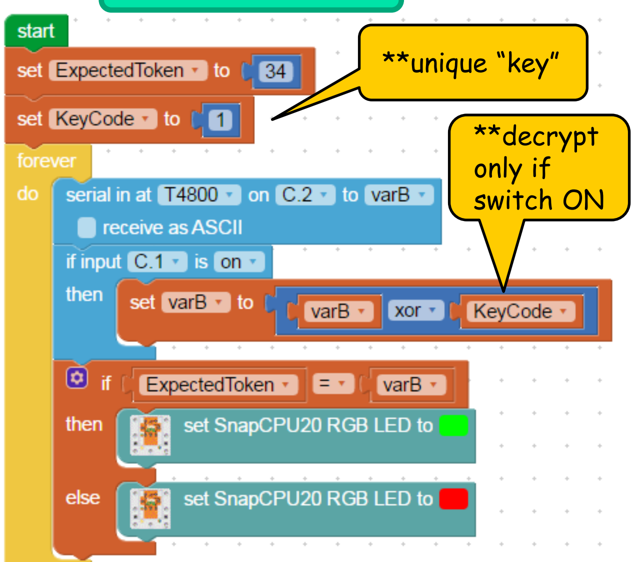
Example Project 4.4b

Receiving a Data Token (8bit packet decryption+MFA)

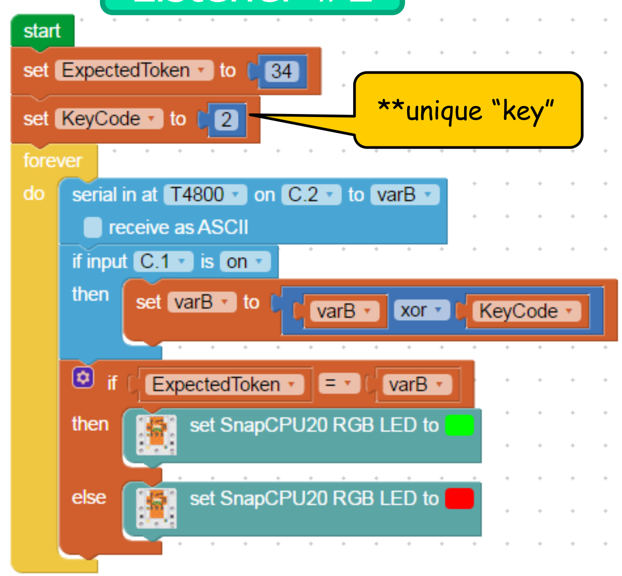
- The “Listener” code is the same as project 4.3b

  Note: since each “Listener” has a unique decryption key; they each get special code
- When powered up, all 3 “Listener” LEDs take turns to glow green and only do so when they receive the specific data packet

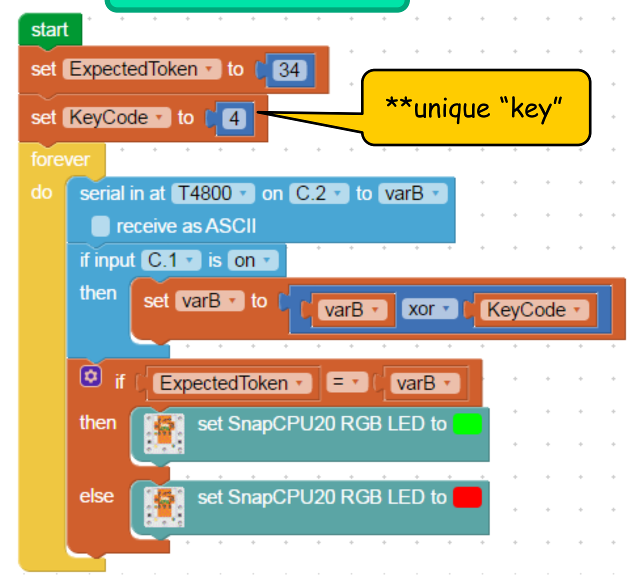
Listener #1



Listener #2



Listener #3



Real World Data Protection

- Project 4.4 represents a simplified version of how the Internet works
- Information is constantly being delivered over the data lines
- The information is delivered in packets, each wrapped in its own key unique to the recipient
- The packet is meaningless without the correct decryption and access keys

Extra Data Challenges

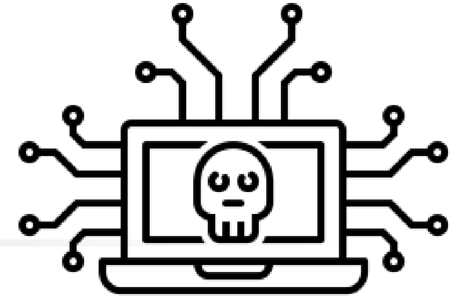
More Hardware Key Encryption

- Using the circuit in project 4.4 replace one of the switch [15] keys with an RFID [61]
- Use the one RFID token's ID number as a software decryption key and send data from the talker using that encryption key
- After 10 packets of data are recieved, reset and wait for a new RFID token swipe

Hint: the RFID key is a number between 0 and 15 ($0F)

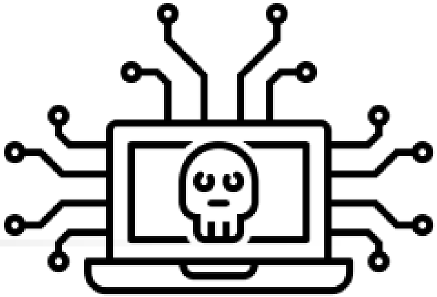
Hint: only read the RFID token once before entering the do…forever loop to listening to the data traffic

Cyber Security

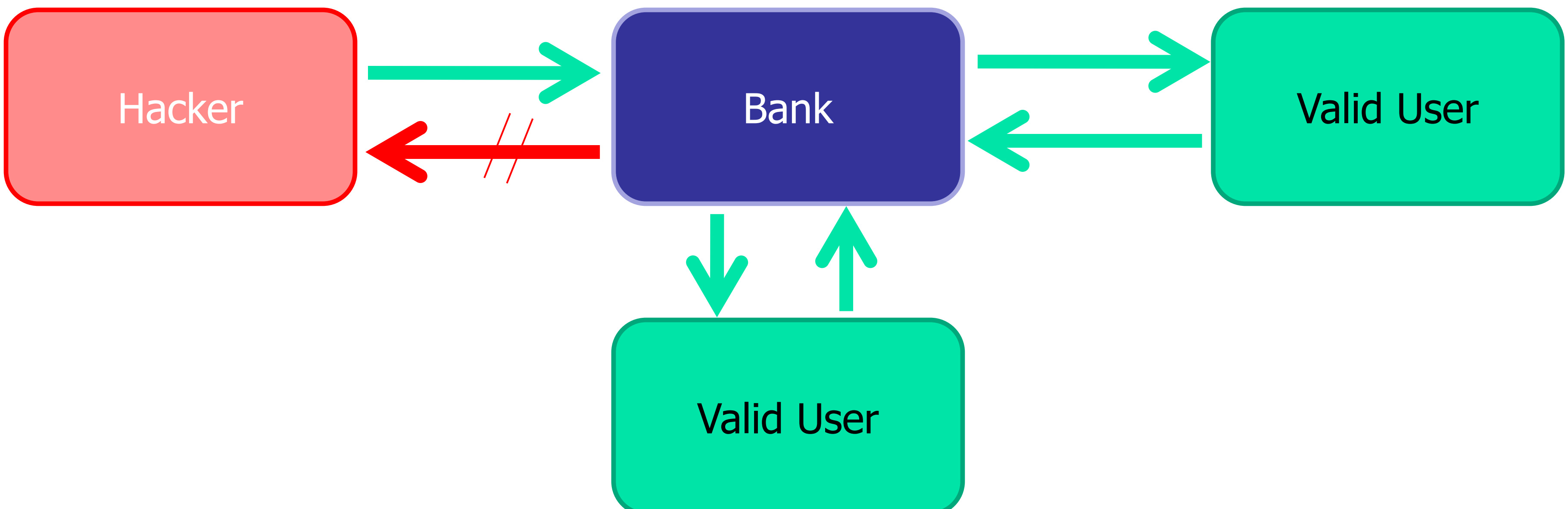


- Cyber security brings together many of the elements discussed already in this project guide for the purpose of protecting networks and data from malicious attack and damage
- For the purpose of demonstration, example projects in this chapter are based on the example attack by a 'hacker' on a 'bank'
  - As the 'hacker' improves their methods the 'bank' improves its cyber security

Example Project Model



- Between valid users the bank will provide free-flowing information validated by a password
- If the bank detects a hacker, it blocks the transfer of information

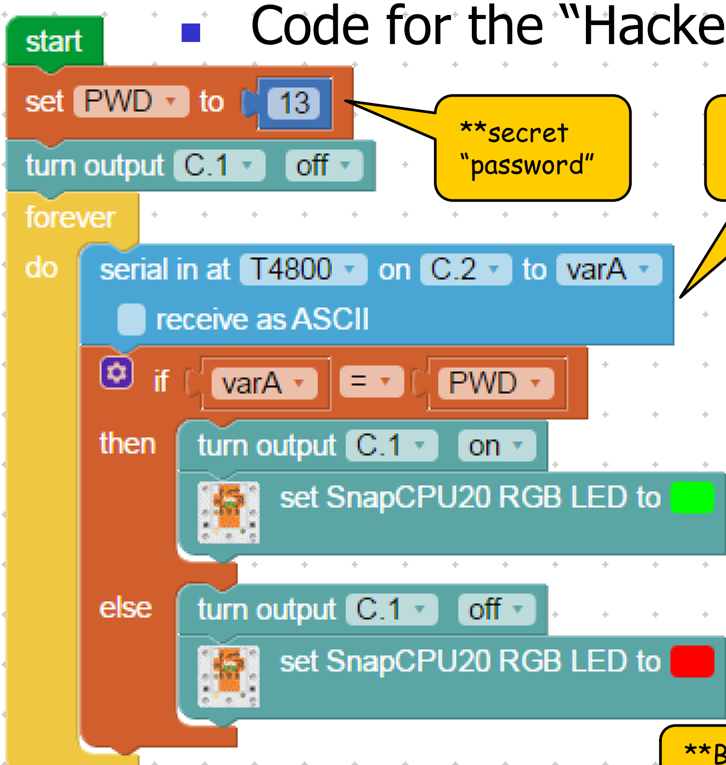


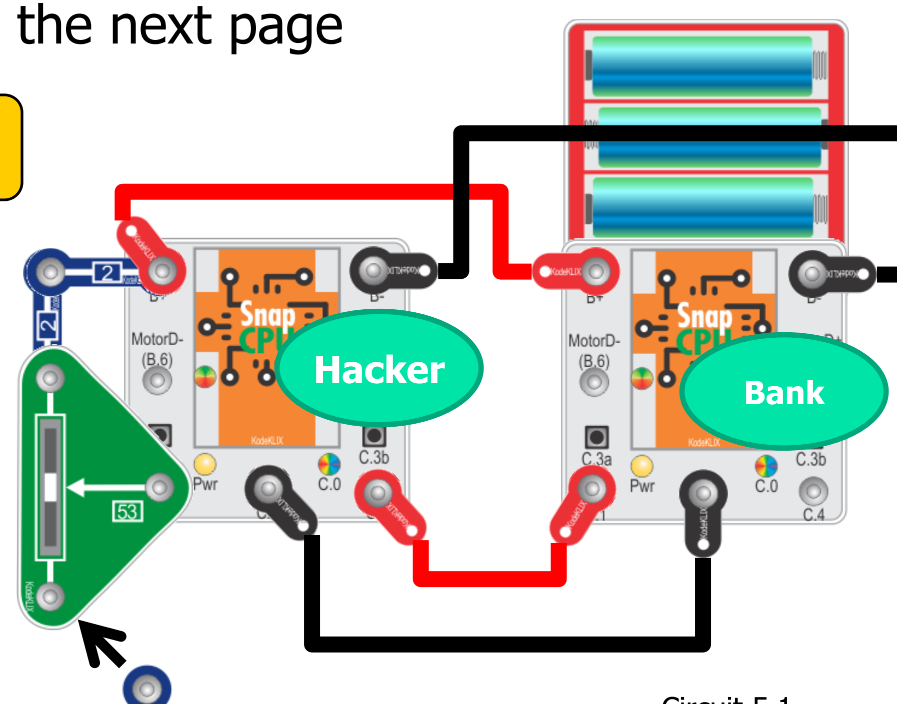
Completed ☐

Example Project 5.1a

Password Spoofing (the Bank)

- Assemble the snap components as shown in circuit 5.1; remember to connect +/- power to each SnapCPU
- Construct the code below and download to the "Bank" SnapCPU
- Code for the "Hacker" is on the next page





Circuit 5.1

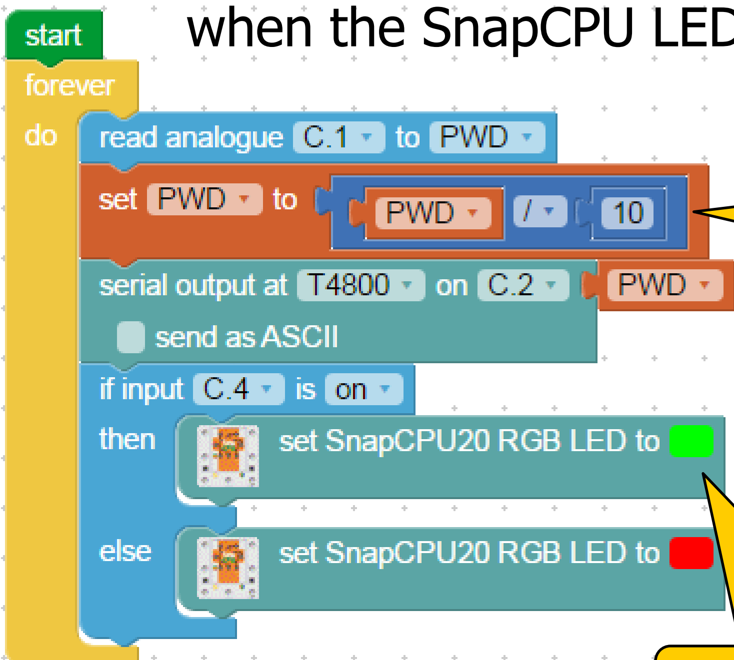
Always remember to power up all SnapCPUs!

Completed

Example Project 5.1b

Password Spoofing (the Hacker)

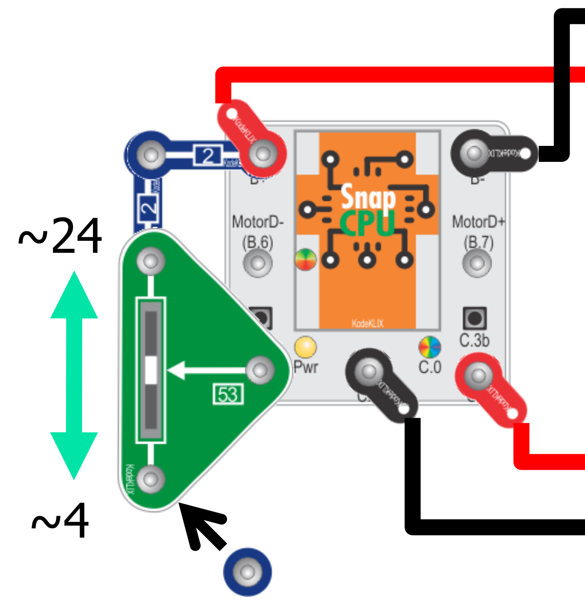
- The Hacker's keyboard is simulated using the varistor [53]; sliding it up or down results in a different codeword being sent to the bank using the serial link
- When the hacker finds the right codeword, the bank is unlocked when the SnapCPU LED changes glows green



**the PWD guess is ranged by dividing by 10 to reduce the possible number of codes available

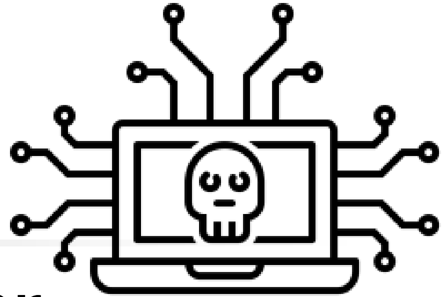
**PWD sent to bank by serial network

**Bank LED glows green if password matched, else glows red



Always remember to power up all SnapCPUs!

Password Spoofing



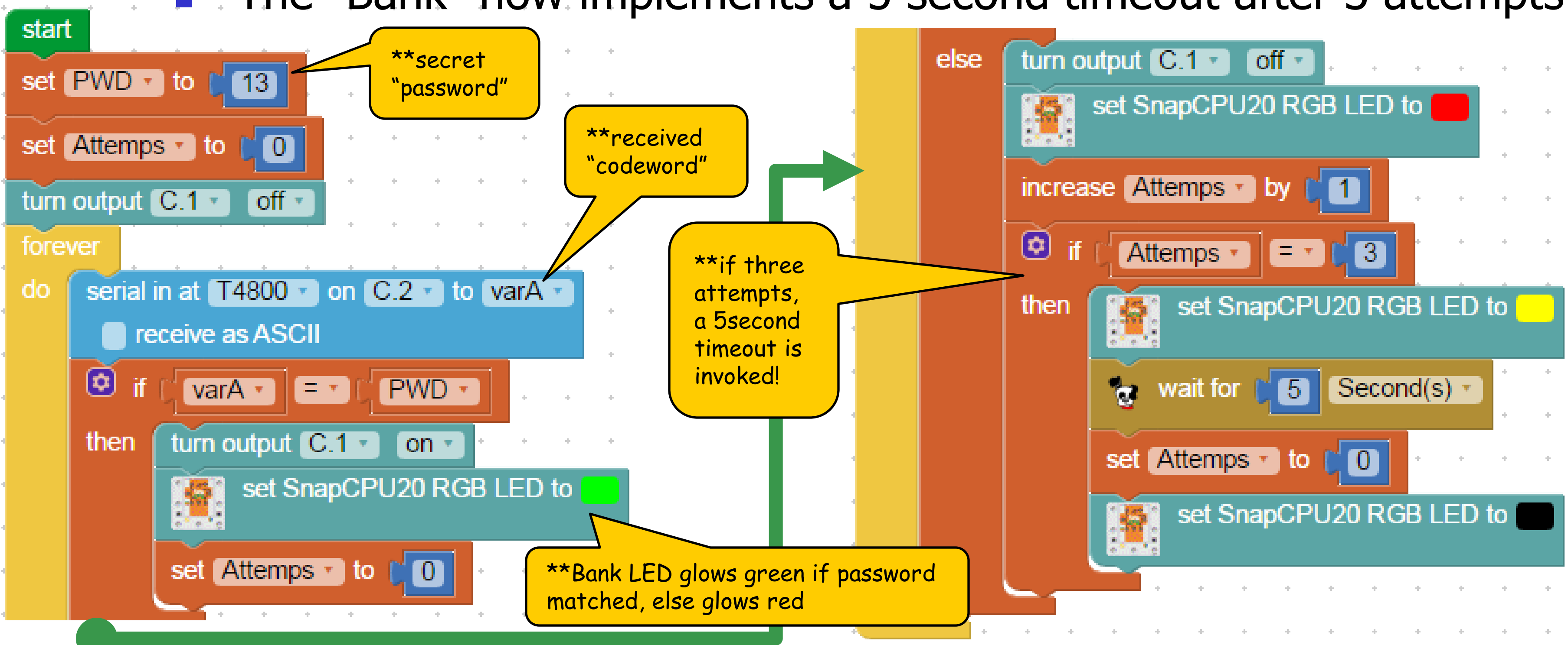
- If cyber security is poor a hacker may guess the password after enough attempts
- In example 5.1, you can easily find the password after sliding the varistor setting
- In example 5.2 the Bank gets wise to this, implementing a 5 second lockout after 3 incorrect password attempts
- The hacker has to try and beat the system!

Completed ☐

Example Project 5.2

Password Spoofing (Bank TimeOut delay)

- Start with assembly of snap components as shown in project circuit 5.1; remember to connect +/- power to each SnapCPU
- Construct the code below and download to the "Bank" SnapCPU
- The "Bank" now implements a 5 second timeout after 3 attempts

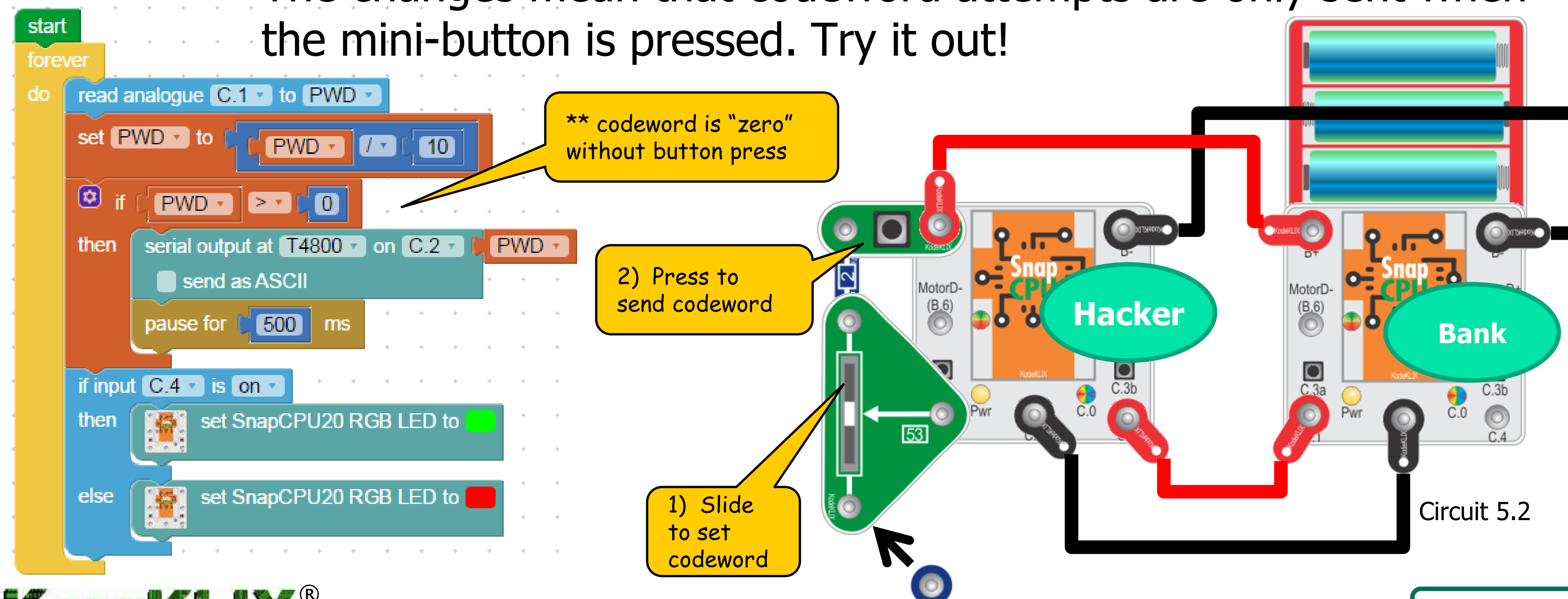


Completed

Example Project 5.3

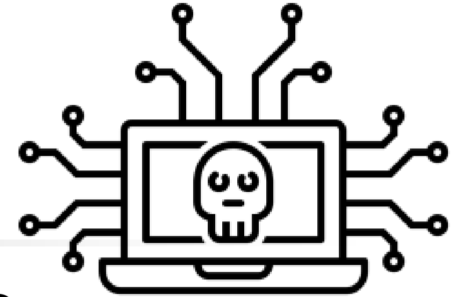
Password Spoofing (Hacker tries again)

- The previous “Hacker” circuit and code sent data so fast, the “Bank” was in constant “Timeout” delay. Time to upgrade!
- Assemble the snap components as shown in circuit 5.2 and download updated code to the “Hacker” SnapCPU
- The changes mean that codeword attempts are only sent when the mini-button is pressed. Try it out!



Circuit 5.2

Brute Force Methods



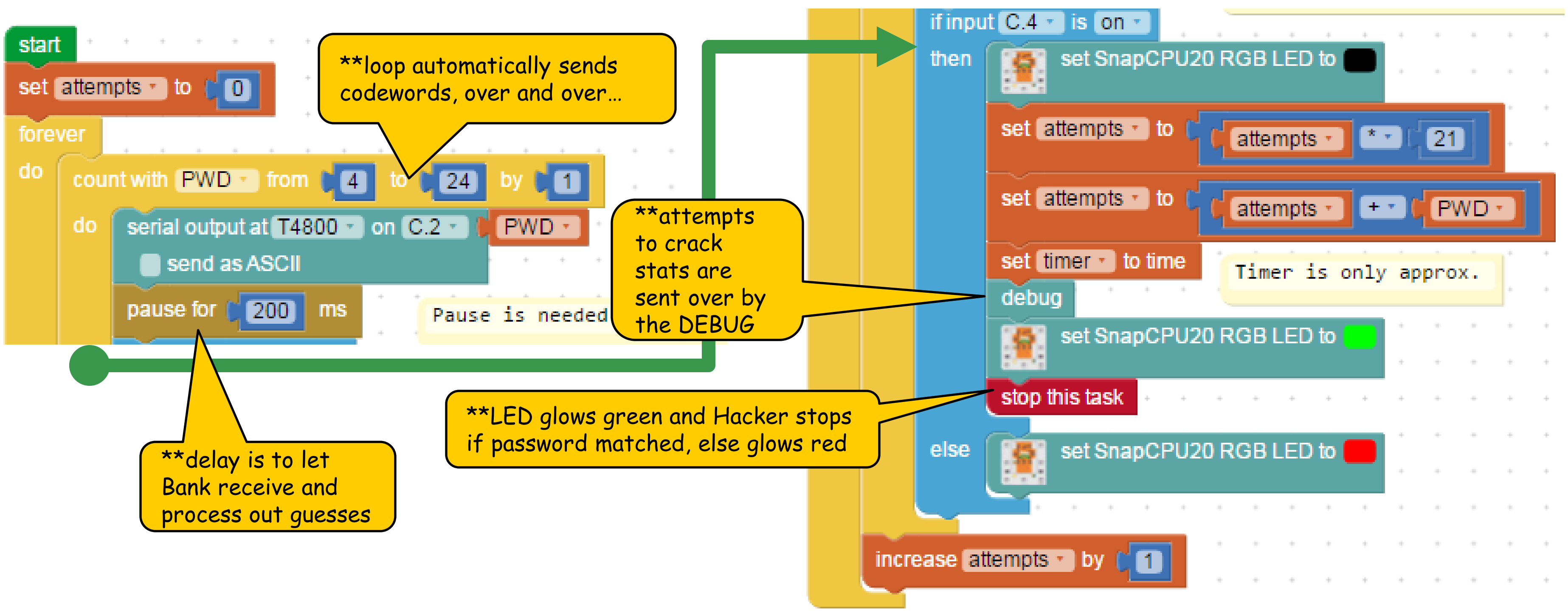
- In project 5.3 the hacker had to manually ‘guess’ the passcode
- A smart hacker would use their computer to do the guessing; usually in a loop until the code gets **cracked**
- This is the simplest **brute force** method of a cyber attack
  - The computer scans for all possible codes
  - Adapts for counter measures, eg lockouts
  - Checks for the successful system access

Completed

Example Project 5.4

Password Spoofing (Hacker gets smarter)

- The "Hacker" is lazy and so writes new code which automatically sends codeword guesses until it succeeds.
- This code update (below) is download to the "Hacker" SnapCPU
- Snap components circuit 5.1 or 5.2 can be used again.

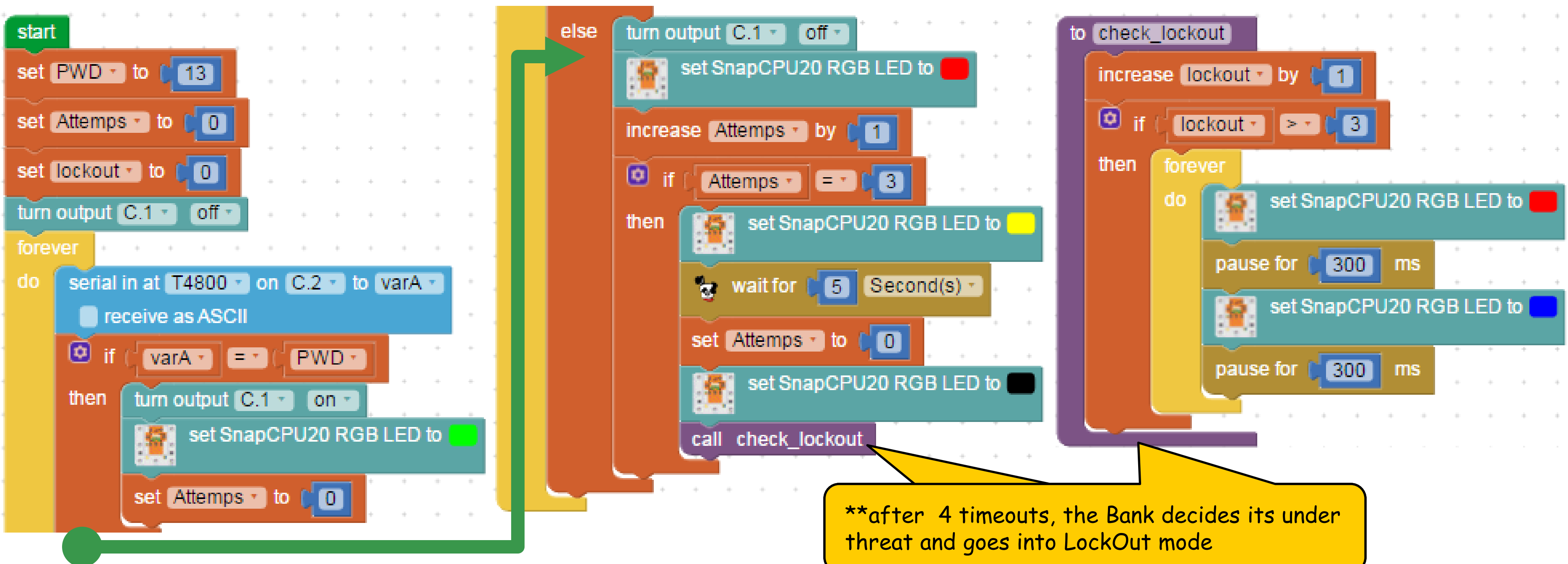


Completed ☐

Example Project 5.5

Password Spoofing (Bank’s last chance)

- The “Bank” upgrades its policy:
  - if 5 second lock-outs are trigged 4-times, then the Bank network is in lockout mode and the police are called.
- Code is updated (below) and downloaded to the “Bank” SnapCPU
- Snap components circuit 5.1 or 5.2 can be used again.



www.kodeklix.com.au

Coding Kits for Kids

Creators:	Nick Coplin, Ellie Coplin
Projects:	www.kodeklix.com.au/snapcpu4stem/
Microchips:	PICAXE
PICAXE:	www.picaxe.com