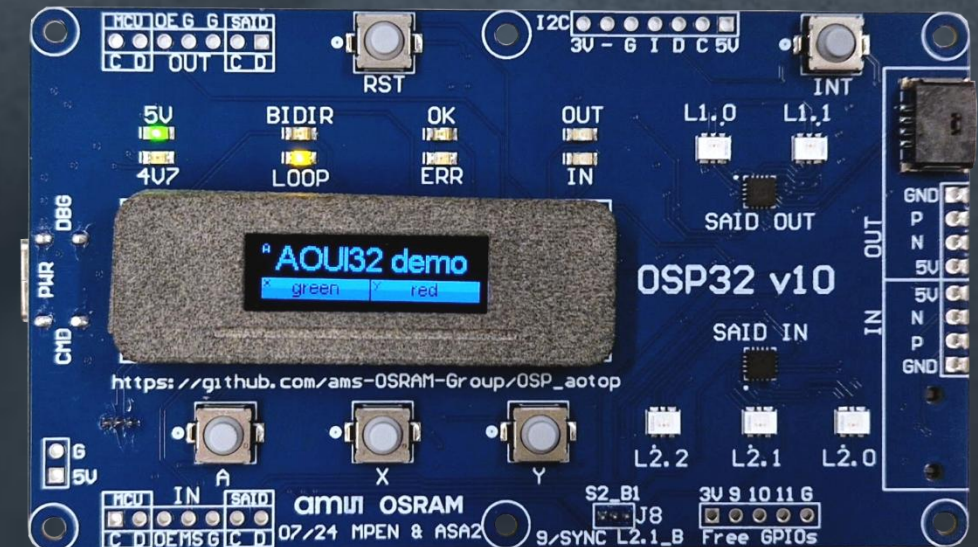


OSP on Arduino – Training – Appendix 1

Using the *Arduino OSP evaluation kit* – Star networks



Sense the power of light

Part A1 – Star networks

Introduction

Hardware

Topologies

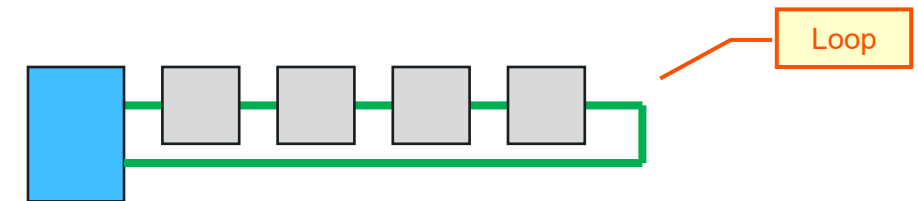
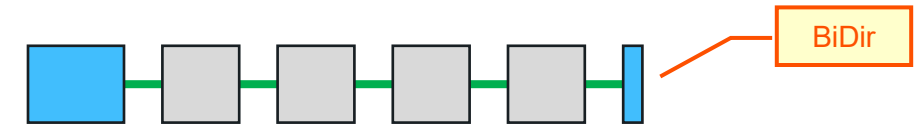
Examples

OSP topologies

Daisy chain

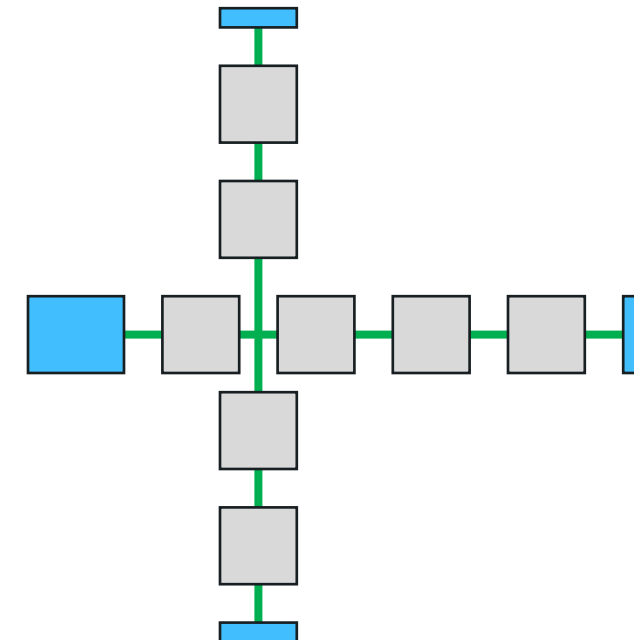
OSP 1.0

- As launched with RGBI's
- Mandates **daisy chaining** of OSP node
- Either in **Bidir** or in **Loop mode**



OSP 1.1

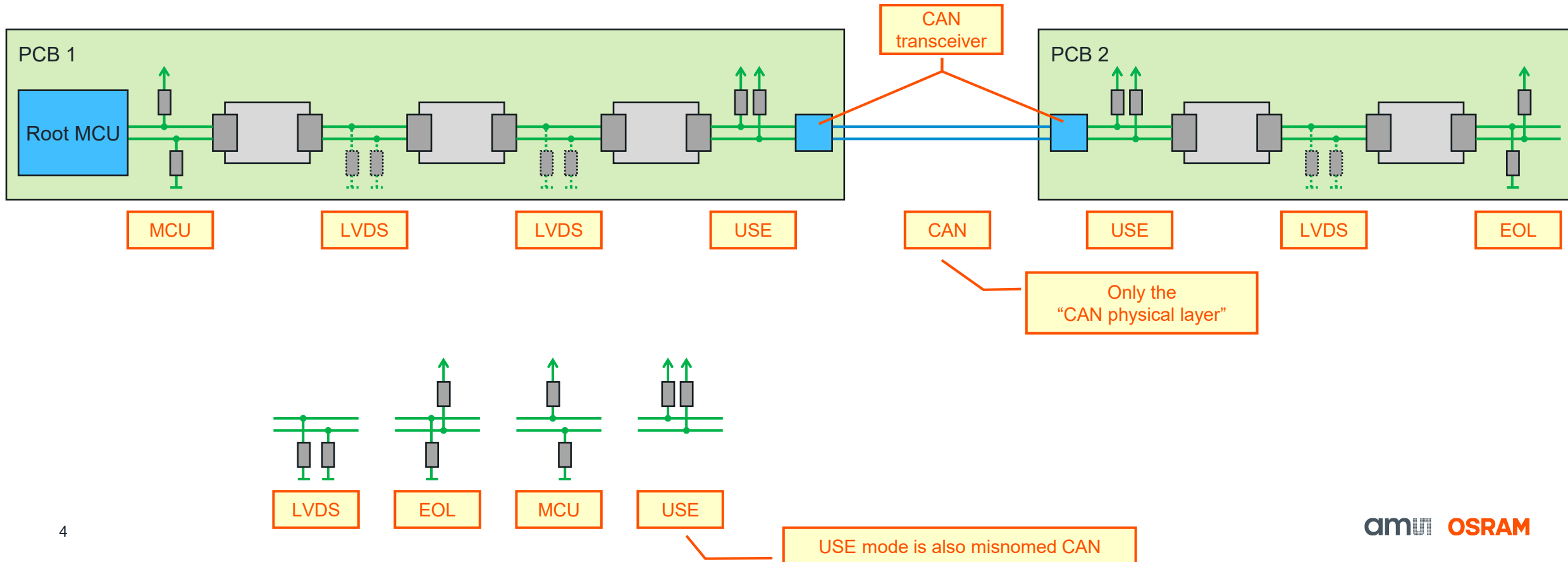
- The SAID offers the feature of **star** network
- Drawn on-purpose in a strange fashion ("star")
- Because the network does not allow for generic branches, i.e. *no* branch from a branch (no *trees*)
- Good news: a SAID *starts* a branch, but the branch can have nodes that do not support branching (like RGBI)



Intermezzo on CAN

Off-PCB connections

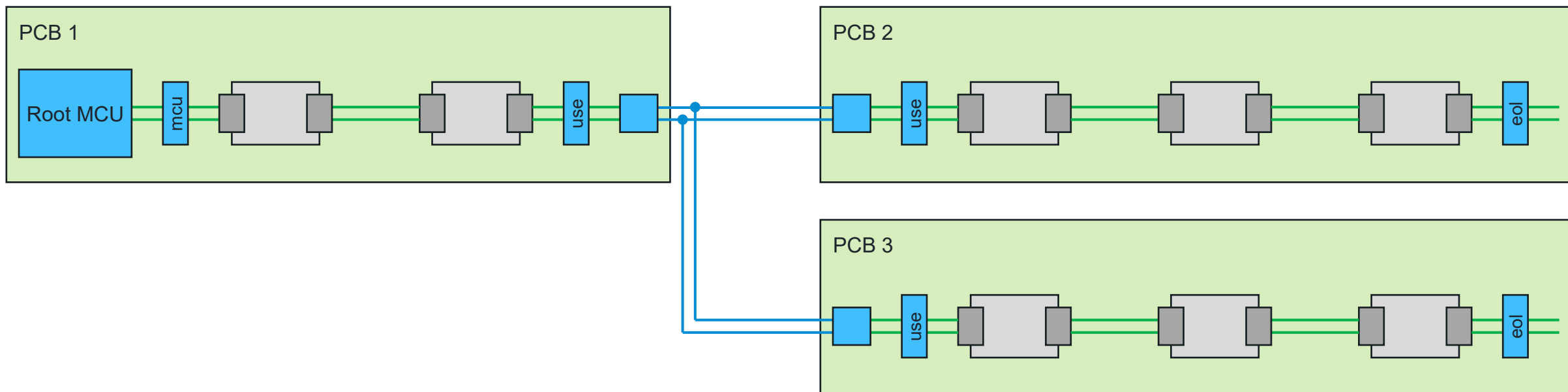
- We saw earlier that SIO ports in a chain are configured for LVDS, except the first (MCU) and the last (EOL)
- There is a fourth flavor: USE (unidirectional single-ended; misnamed CAN in some documentation)
- A standard component (a CAN transceiver) translates USE to *CAN physical layer* for more robust off-PCB connections



Using CAN for branching

Branch in CAN domain

- Splitting branches off the OSP chain is best done in the CAN domain
- Splitting LVDS causes signal degradation



- (Technically it is possible to run branches in loop back mode, but we will skip that in this presentation)

Sense the power of light

Part A1 – Star networks

Introduction

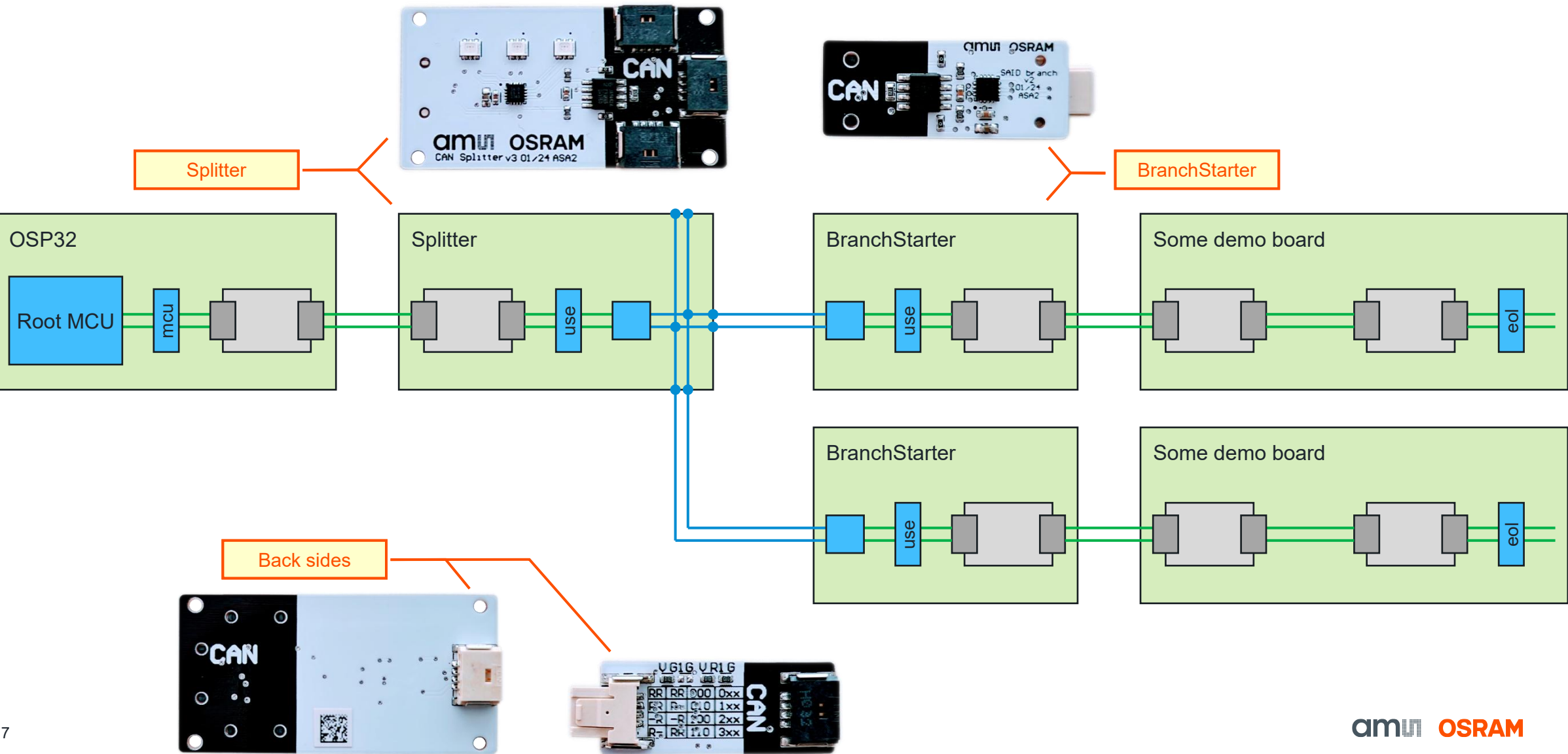
Hardware

Topologies

Examples

Two modules added: *Splitter* and *BranchStarter*

Eval kit is a modular system



Looking at the Splitter

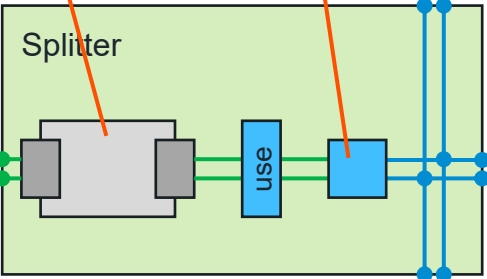
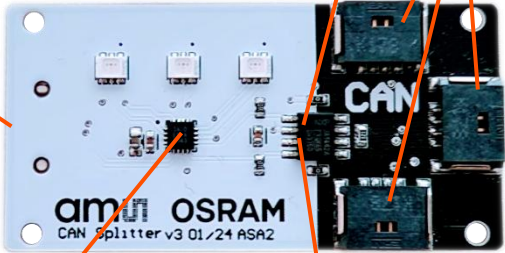
Eval kit is a modular system

Why is there a SAID on the splitter board?

LVDS port ("IN")

USE-CAN

3× CAN port ("OUT")



Splitter

- Contains the USE to CAN transceiver
- Contains 1 LVDS (input) port
- Contains 3 CAN (output) ports (arbitrary choice for 3)
- Contains a SAID – why?
 - With 3 RGB triplets (just because)

SAID

- The SAID is *not* needed
- It has one small feature
 - In the OTP, bit 1E.6 is known as BRANCH_POINT
 - It is reserved by the eval kit
 - It identifies a SAID that splits the OSP chain
 - Purpose: enable the auto detect of the chain topology “is there a splitter” and “where is the splitter?”

New location
(was 0E.7)

OTP address	Bits							
	7	6	5	4	3	2	1	0
0D	CH_clustering[2:0]		haptic_driver		SPI_mode	sync_pin_en	star_net_en	i2c bridge_en
0E					otp_add_en	star_net_otp_addr[2:0]		
0F								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
1A								
1B								
1C								
1D								
1E	Cust_lock							
1F	CRC2<7:0> (could also be used for other data)							

customer area

Looking at the BranchStarter

Eval kit is a modular system

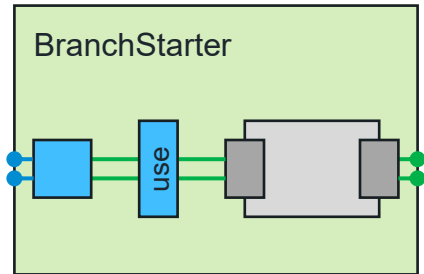
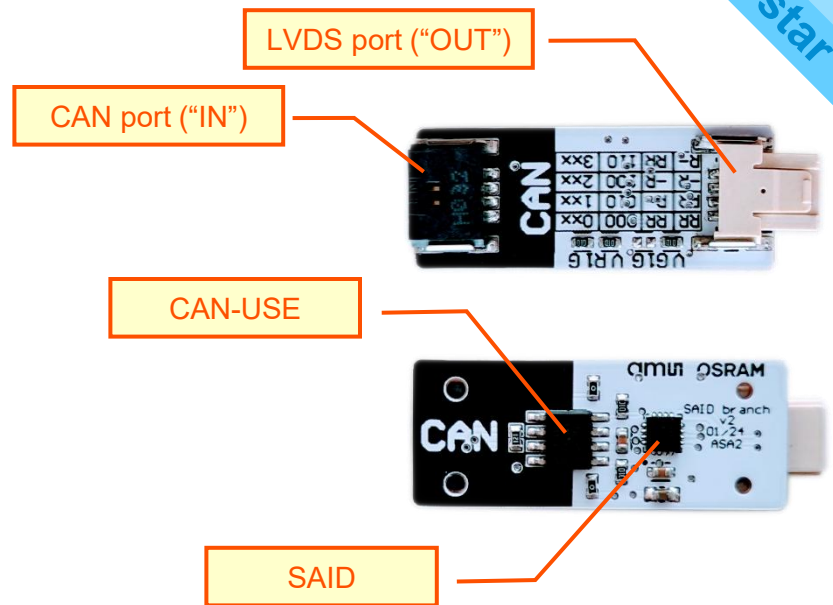
BranchStarter

- Contains CAN (input) port
- Contains LVDS (output) port
- Contains CAN to USE transceiver
- Contains a SAID – why?

USE to CAN transceiver
is same as
CAN to USE transceiver
(it is a symmetrical device)
“input” and “output” is a helpful but limited view

SAID

- The SAID implements the key feature: the (a) branching mechanism
- Implementation:
 - SAID implements a telegram filter
 - The SAID has a 3-bit mask
 - There is an OTP bit enabling filtering
- SAID lets pass a telegram with
 - a broadcast address (000)
 - a groupcast address (3F0–3FE)
 - a unicast address, only if upper 3 bits (MSB) equal the mask



address	cast
000	broadcast
001 – 3EF	1007 unicast addresses
3F0 – 3FE	15 group addresses
3FF	reserved

Branch mask

Two ways to set the mask

Notation for mask

- Write in binary 0b100XXXXXXX (correct but long)
- Write in hex 0x2XX (not completely correct – MSB of first X)
- or as 0x000, 0x080, 0x100, 0x180, 0x200, 0x280, 0x300, 0x380 (start address of each branch)

There are two ways to configure the mask

- In both cases: enable filtering (**star_net_en**)
- Either enable **otp_addr_en**
 - Now the **mask comes from OTP (star_net_otp_addr)**
- Or disable **otp_addr_en** (as done on the BranchStarter PCB)
 - Now **mask comes from pads G1 and R1**
 - By adding pull-up or pull-down or float
 - At the back of PCB is a table (subset of Table 9 in datasheet)

9	8	7	6	5	4	3	2	1	0
Mask 2	Mask 1	Mask 0							
Nibble 2			Nibble 1			Nibble 0			

9	8	7	6	5	4	3	2	1	0
1	0	0	x	x	x	x	x	x	x
2			x			x			

Actual pattern: R- and RR

Lookup

Bin mask is 110 addresses 0b110XXXXXXX pass

Hex mask: 0x3XX pass ("branch 3")

Mask via OTP (or pads)

Enable filtering

Mask in OTP

0D	CH_clustering[2:0]			haptic_driver	SPI_mode	sync_pin_en	star_net_en	i2c_bridge_en	customer area
0E					otp_addr_en		star_net_otp_addr[2:0]		
0F									
10									

Sense the power of light

Part A1 – Star networks

Introduction

Hardware

Topologies

Examples

Intermezzo on initbidir

Subtle behavior details

The initbidir (and initloop) have subtle behavior

Specification of *OSP*, not specific to *SAID*

Open System Protocol 1.0

A device receives the INIT-command with its new address set in the address field. It then sets its own address to the new address, increments the value by one and sends the INIT-command to the next unit in the chain. The last unit in the chain returns its own address to the master and stops the initialization procedure. The response is either sent in backward or forward direction depending on the command flavor used.

Note: An initialized node receiving an INIT-command with a target address different from its own will fast-forward the INIT-command without incrementing the address and without interpreting the command.

When an init telegram arrives at a node, with destination address A

(1) Process command

- If the node is uninitialized: use A as address, step A
- If the node is initialized and address=A: step A
- If the node is initialized and address≠A: skip – but still do (2)

Normal case: ensures that in a “reset” chain all nodes get consecutive addresses

Repair case: ensures that a second identical init telegram leaves all as-is except nodes that had PoR

Branch case: ensures that a second init telegram with other base address skips all init nodes

(2) Send telegram out

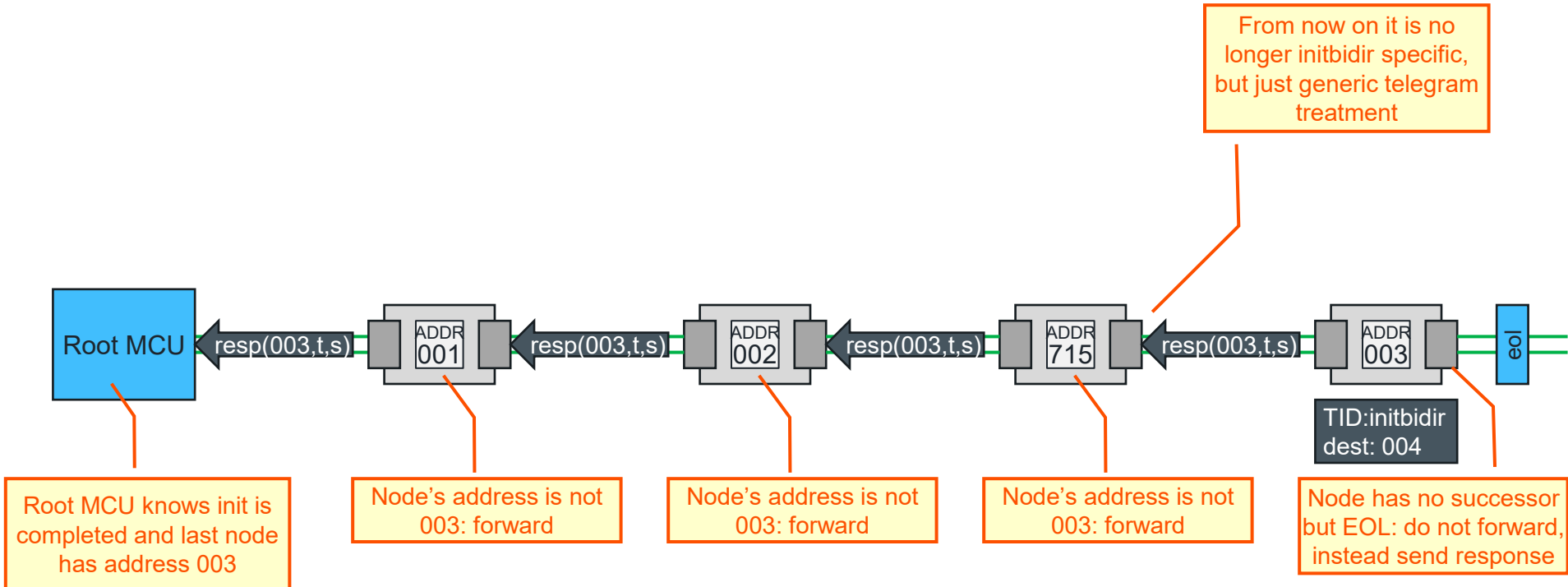
- If the node has successor: forward telegram (with new A)
- If the node has EOL: send response telegram with node's status

Bug: when a SAID is already initialized a new init telegram (with a different address) causes a response with a CRC error



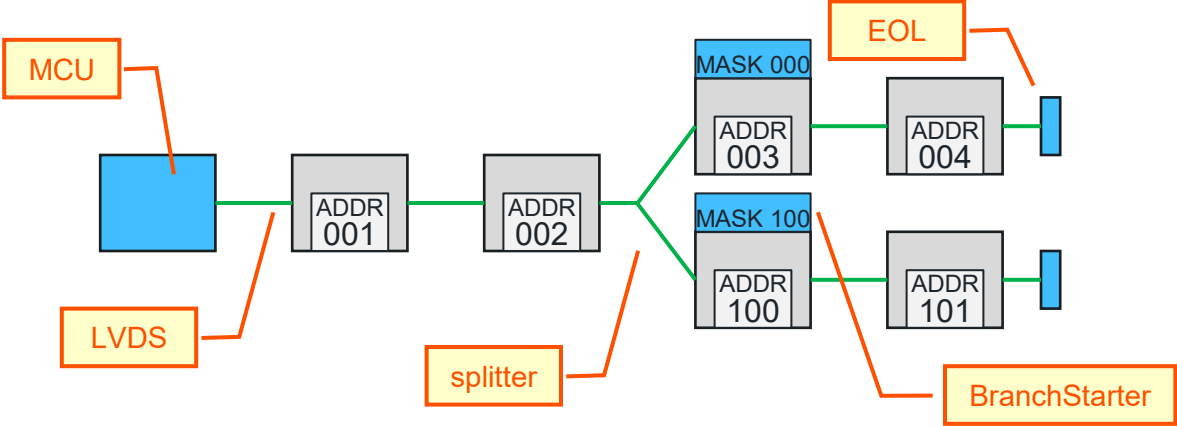
Intermezzo on initbidir

Seeing the subtleties



Branching topologies

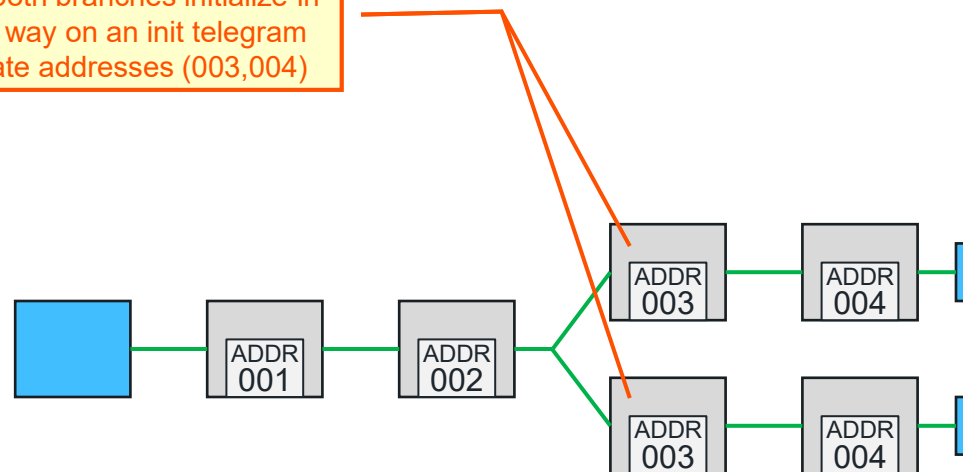
Simplified drawing – legend



Branching topologies – Wrong

Splitter

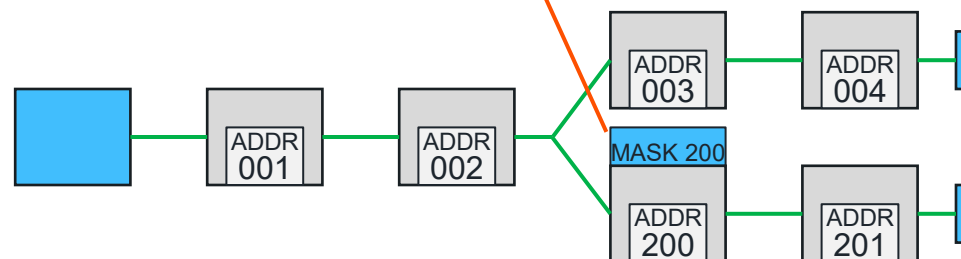
Splitter without BranchStarter does not work; both branches initialize in the same way on an init telegram → duplicate addresses (003,004)



Branching topologies – Wrong

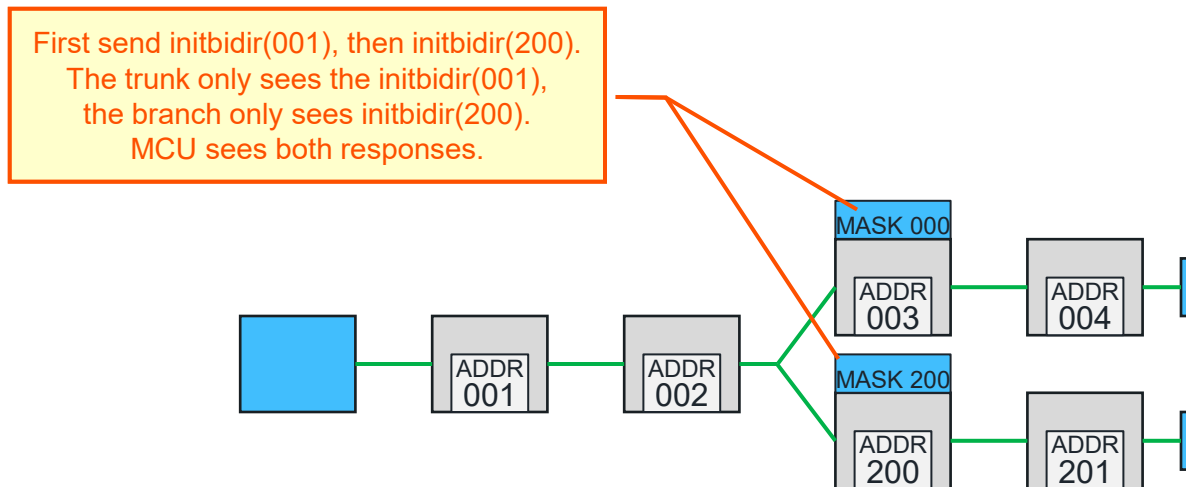
Splitter plus BranchStarter for branch (not for trunk)

First send initbidir(001), then initbidir(200).
In branch, the filter stops the first init, but allows the second init.
Problem: second init causes response from *both* branches.
They collide at node 002, MCU doesn't see 2nd response.



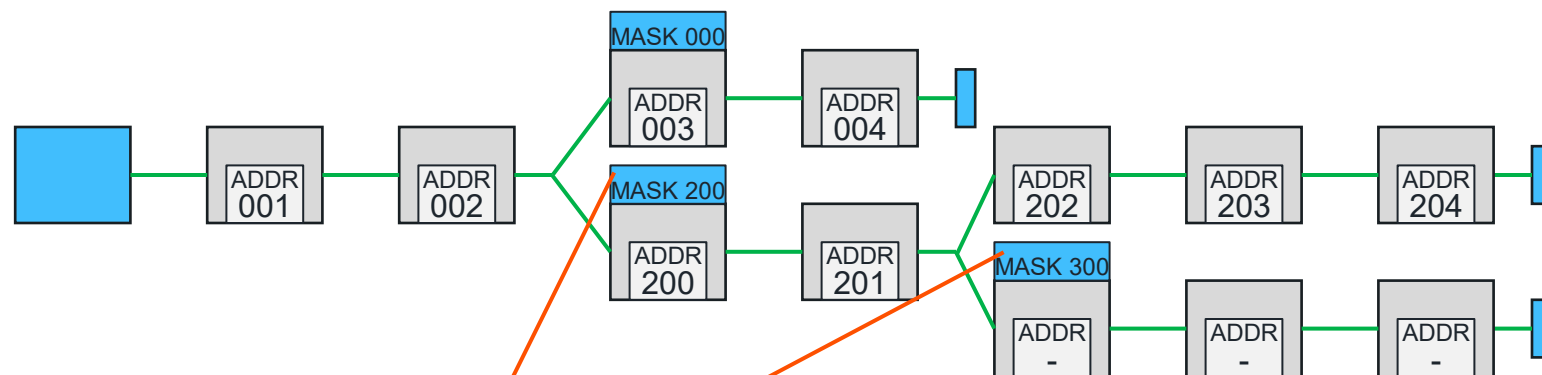
Branching topologies – Success

Splitter plus BranchStarter for branch and trunk



Branching topologies – Wrong

Cascaded filters

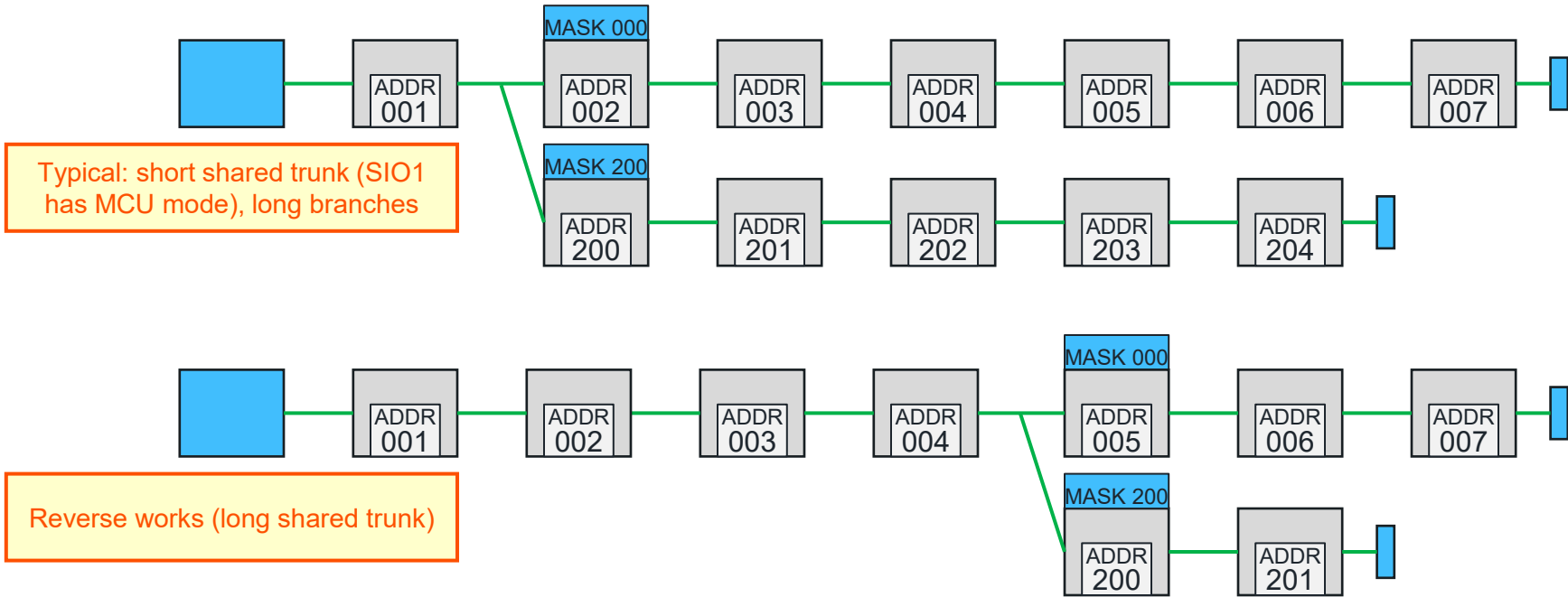


Two filters in a row, the 300 branch does not receive any (unicast) telegram

Branching topologies – simple

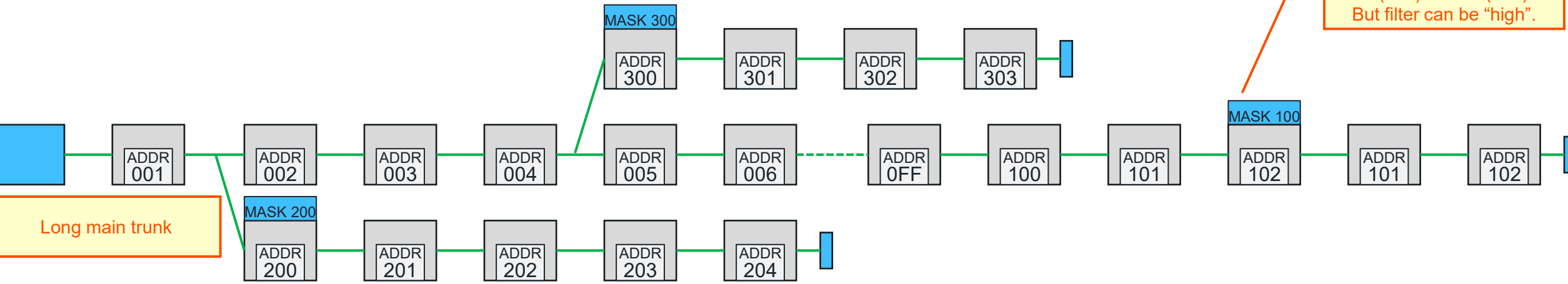
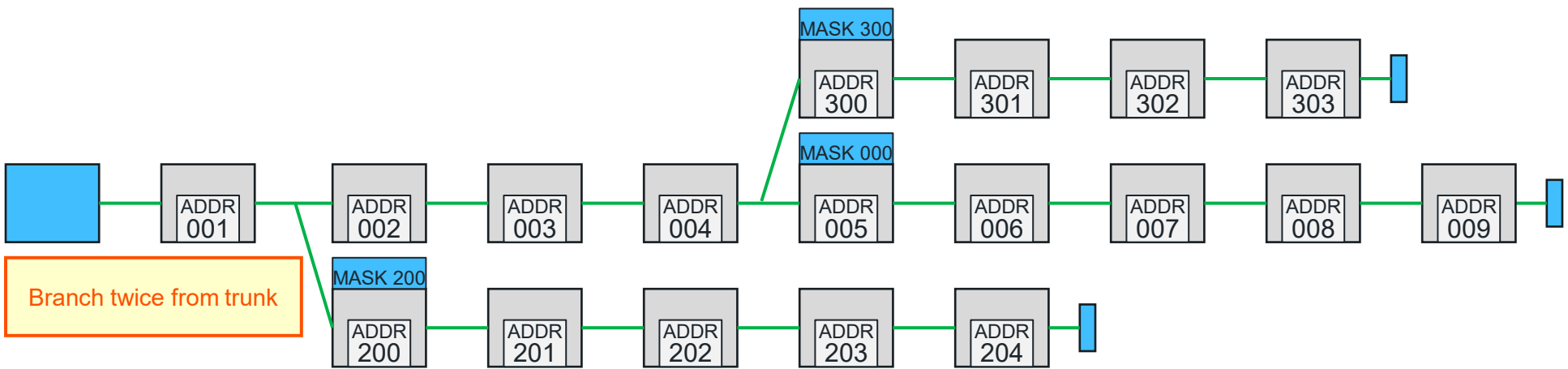
Splitter plus BranchStarter for all branches (including trunk)

Note trunk does not have to be branch 0x0XX; trunk is the init(0xtXX) that comes first after reset (but convention dictates 0x0XX)



Branching topologies - advanced

Splitter plus BranchStarter for all branches (including trunk)



Branching rule

Definitions

- Every branch ends in a *leaf*.
- We say an *OSP tree has branches* when the number of leaves is greater than 1 (if it is 1, the tree only consists of the trunk)
- Every leaf is connected to the RootMCU by a unique path, known as the *root path*.

When an OSP tree has branches, every root path must have exactly one BranchStarter.

- More than one means double filter, leaf will not receive any (unicast) telegram.
- Less than one means possible clashes for response telegrams (from serial cast commands)

Sense the power of light

Part A1 – Star networks

Introduction

Hardware

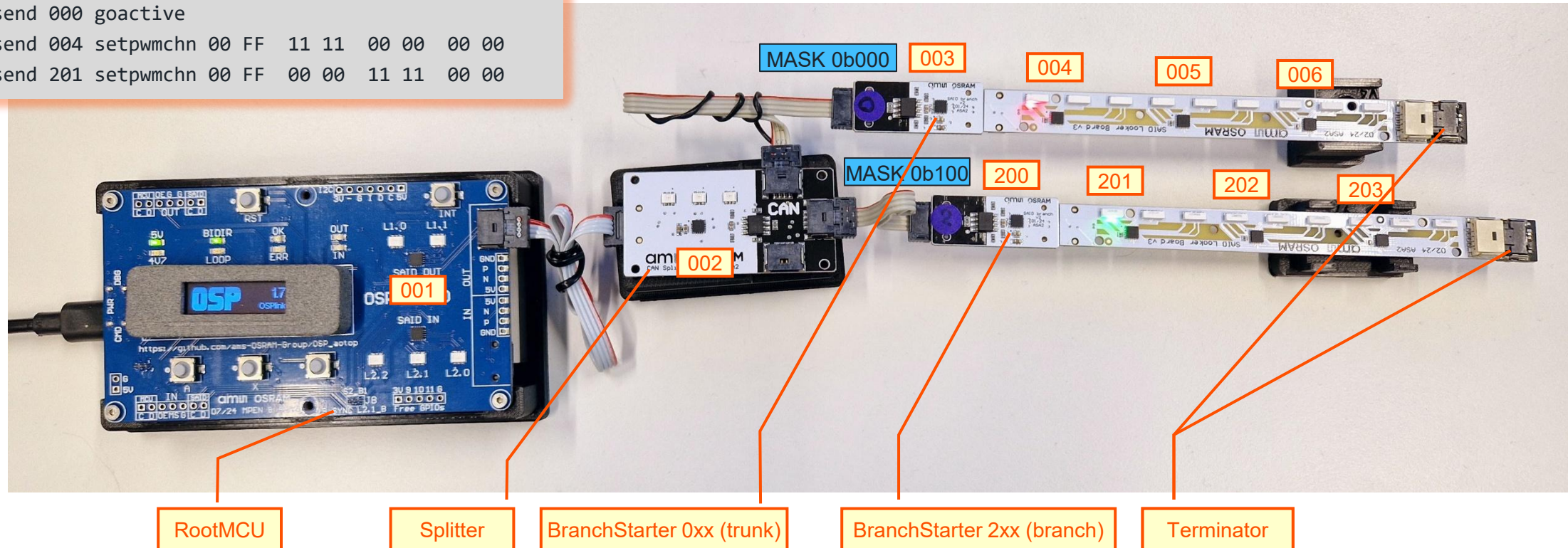
Topologies

Examples

Demonstrating simple branching

Using OSPLink (from eval kit)

```
>> osp dirmux bidir
>> osp send 000 reset
>> osp send 001 initbidir
>> osp send 200 initbidir
>> osp send 000 clrerror
>> osp send 000 goactive
>> osp send 004 setpwmchn 00 FF 11 11 00 00 00 00
>> osp send 201 setpwmchn 00 FF 00 00 11 11 00 00
```



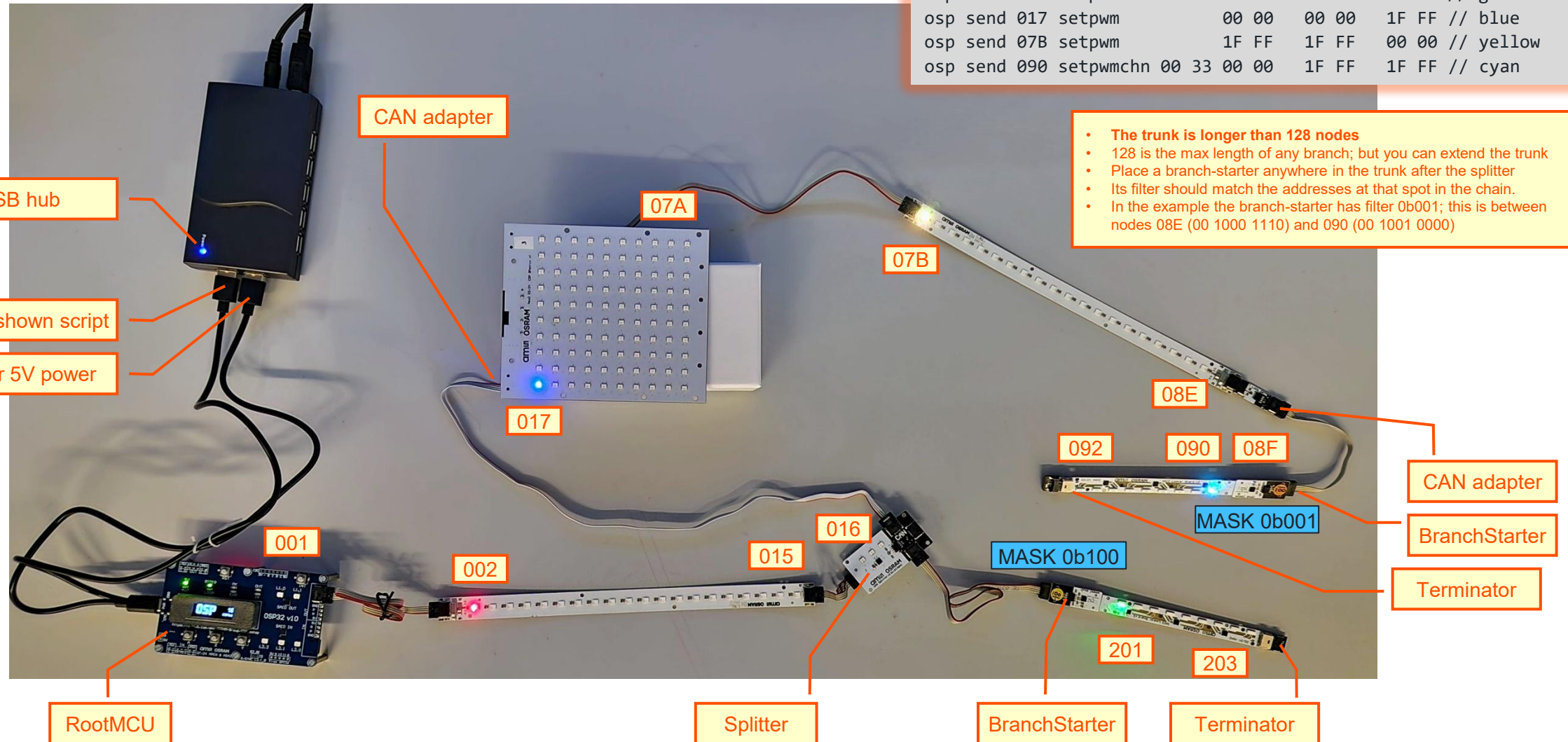
Using OSPlink (from eval kit)

```

osp send 000 reset
osp send 001 initbidir
osp send 200 initbidir
osp send 000 clrerror
osp send 000 goactive
osp send 002 setpwm          1F FF    00 00    00 00 // red
osp send 201 setpwmchn 00 33 00 00    1F FF    00 00 // green
osp send 017 setpwm          00 00    00 00    1F FF // blue
osp send 07B setpwm          1F FF    1F FF    00 00 // yellow
osp send 090 setpwmchn 00 33 00 00    1F FF    1F FF // cyan

```

- **The trunk is longer than 128 nodes**
- 128 is the max length of any branch; but you can extend the trunk
- Place a branch-starter anywhere in the trunk after the splitter
- Its filter should match the addresses at that spot in the chain.
- In the example the branch-starter has filter 0b001; this is between nodes 08E (00 1000 1110) and 090 (00 1001 0000)



am  OSRAM

Case study — part 1: doing it wrong

Simple example done wrong, then corrected

Hardware setup 1

Wrong: CAN adapter

OSP32 board
flashed with
SAIDdemo

CAN splitter

CAN adapter
(not a BranchStarter)
This is wrong

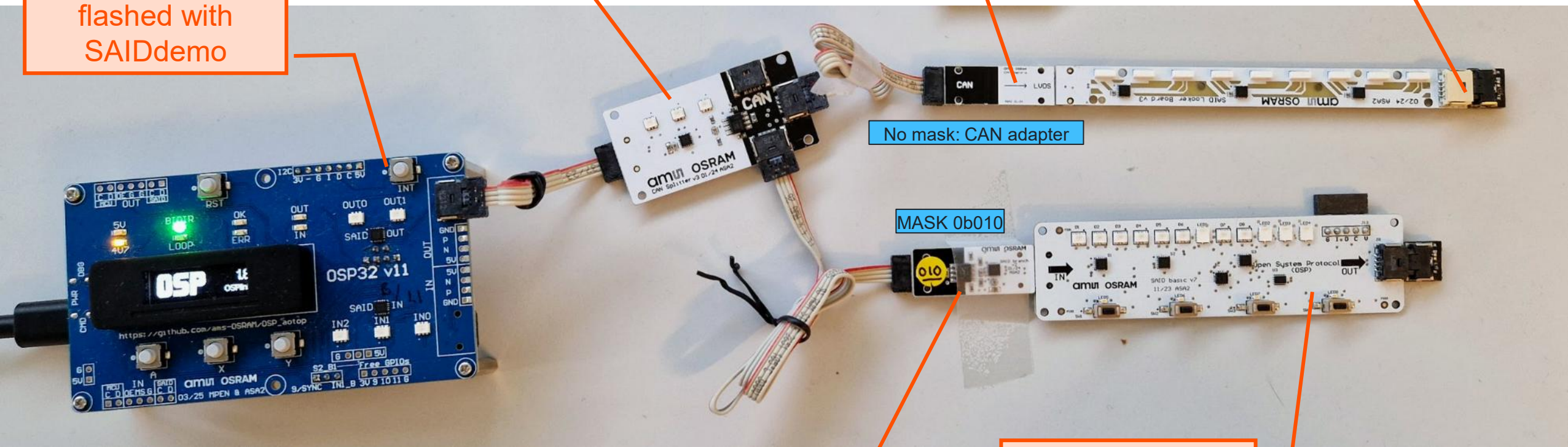
SAIDlooker
(then terminator)

No mask: CAN adapter

MASK 0b010

BranchStarter

SAIDbasic
(then terminator)



Observations

On the hardware

- The CAN splitter
 - Changes physical layer from LVDS to CAN
 - Has three ERNI connectors in parallel in the CAN domain
- The side branch (bottom one in photo) uses a *BranchStarter*
 - Changes physical layer back from CAN to LVDS
 - Has a SAID that implements address filtering (here 0b010, so addresses 0x1XX)
- The trunk (main branch) uses a *CAN adapter*
 - Changes physical layer back from CAN to LVDS
 - Has no address filtering
 - This is wrong

Software experiment

Try to initialize the chain

We are using OSPlink to test this configuration

- “osp send 000 reset”
broadcasts a reset command
- “osp send 001 initbidir”
initializes the trunk
- “osp send 101 initbidir”
initializes the “0x1XX” (0b010) branch
- Seems ok, but when we draw the response,
“osp fields A4 05 02 6F 50 91”
we see a **CRC error**

Reason:

- both branches see the 101 initbidir
- both respond, responses may clash

The last trunk node gets confused when it is initialized twice and sends the bogus response we see here

OSPlink

OSPlink - version 1.8

```
splink: init(MCU-B)
osp: init
cmd: init
mw: init
ui32: init

No 'boot.cmd' file available to execute
Type 'help' for help
>> osp send 000 reset
tx A0 00 00 22
rx none ok
>> osp send 001 initbidir
tx A0 04 02 A9
rx A0 15 02 00 50 AA (147 us) ok
>> osp send 101 initbidir
tx A4 04 02 91
rx A4 05 02 6F 50 91 (117 us) ok
>>
>> osp fields A4 05 02 6F 50 91
```

A4		05		02		6F		50		91	
1 0 1 0 0 1 0 0		0 0 0 0 0 1 0 1		0 0 0 0 0 0 1 0		0 1 1 0 1 1 1 1		0 1 0 1 0 0 0 0		1 0 0 1 0 0 0 1	
+-----+		+-----+		+-----+		+-----+		+-----+		+-----+	
preamb1		address		psi		command		payload		payload	
+-----+		+-----+		+-----+		+-----+		+-----+		+-----+	
0xA		0x101		0x2		0x02		0x6F		0x50	
-		unicast(257)		2		initbidir		111		80	
+-----+		+-----+		+-----+		+-----+		+-----+		+-----+	
										0x91 (ERR) 0xAC	
										145 (ERR) 172	
+-----+		+-----+		+-----+		+-----+		+-----+		+-----+	

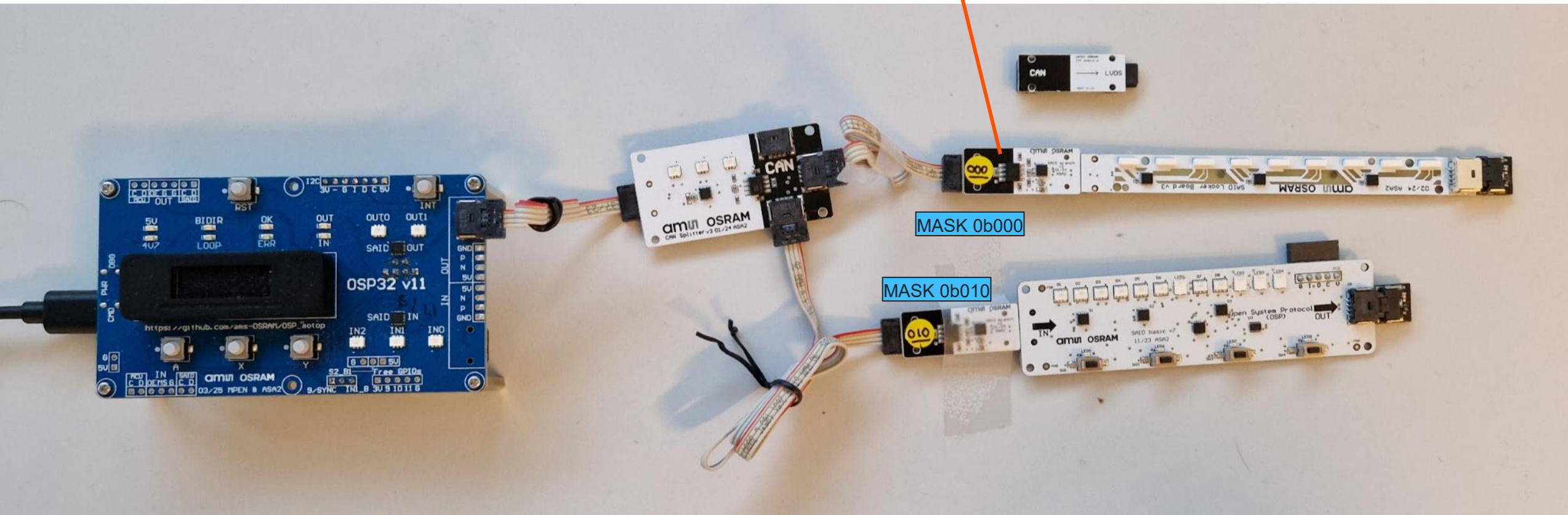
```
>>
```

Case study – part 2: Fix by adding filter on trunk

Simple example done wrong, then corrected

Correct: BranchStarter on trunk

BranchStarter
(with 0b000 as filter)



Software experiment

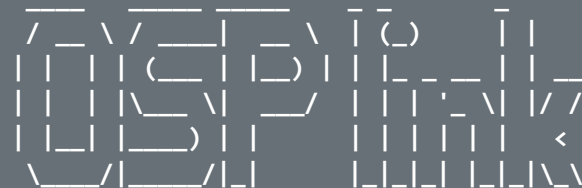
Initialize both branches

We are using OSPlink to test this configuration

- “osp send 000 reset”
broadcasts a reset command
- “osp send 001 initbidir”
initializes the trunk
- “osp send 101 initbidir”
initializes the “0x1XX” (0b010) branch
- Now is ok, “osp fields A4 21 02 00 40 FE”
we see a correct CRC and 7 nodes
(on SAIDbasic)

Why does it work:

- The trunk now also has a filter
- Trunk no longer sees initbidir for side branch
- Trunk no longer (also) responds to that tele



OSPlink - version 1.8

```
spi: init(MCU-B)
osp: init
cmd: init
mw: init
ui32: init
```

No 'boot.cmd' file available to execute

Type 'help' for help

```
>> osp send 000 reset
```

```
tx A0 00 00 22
```

```
rx none ok
```

```
>> osp send 001 initbidir
```

```
tx A0 04 02 A9
```

```
rx A0 19 02 00 50 1B (171 us) ok
```

```
>> osp send 101 initbidir
```

```
tx A4 04 02 91
```

```
rx A4 21 02 00 40 FE (247 us) ok
```

```
>> osp fields A4 21 02 00 40 FE
```

+-----+-----+-----+-----+-----+-----+-----+													
A4		21		02		00		40		FE			
1 0 1 0 0 1 0 0		0 0 1 0 0 0 0 1		0 0 0 0 0 0 1 0		0 0 0 0 0 0 0 0		0 1 0 0 0 0 0 0		1 1 1 1 1 1 1 0			
+-----+-----+-----+-----+-----+-----+-----+													
preamble		address		psi		command		payload		payload		crc	
+-----+-----+-----+-----+-----+-----+-----+													
0xA		0x108		0x2		0x02		0x00		0x40		0xFE (ok)	
-		unicast(264)		2		initbidir		0		64		254 (ok)	
+-----+-----+-----+-----+-----+-----+-----+													

```
>>
```

Case study — part 3: switch LED in both branches

Simple example done wrong, then corrected

Switching on the last LED in both branches

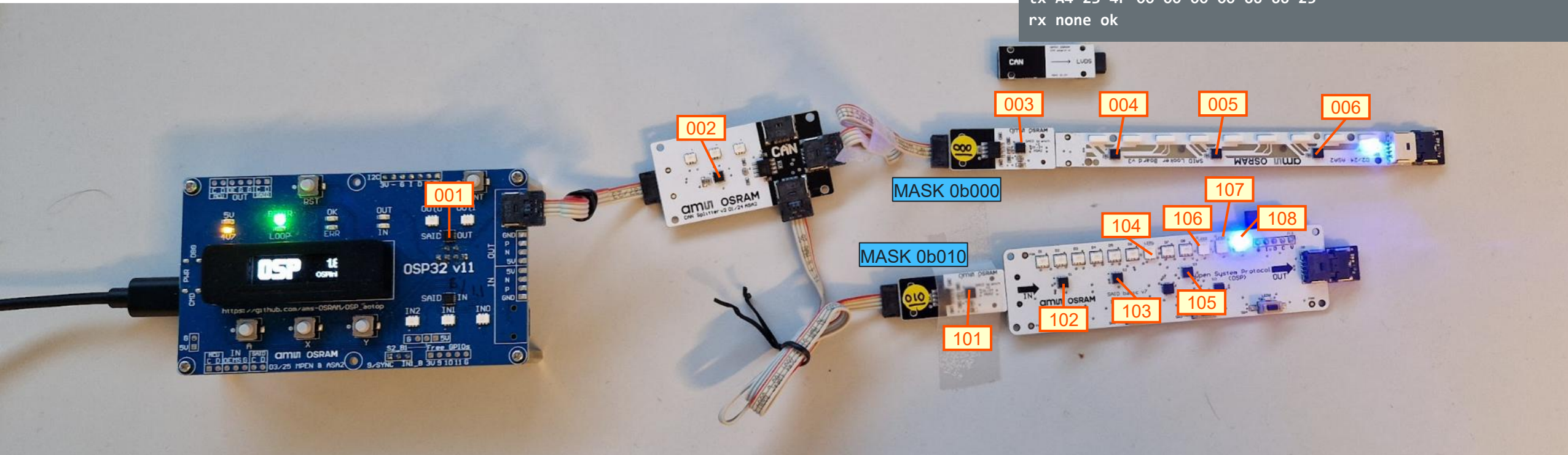
Demoing initbidir was successful

```
>> osp send 000 clrerror
tx A0 00 01 0D
rx none ok

>> osp send 000 goactive
tx A0 00 05 B1
rx none ok

>> osp send 006 setpwmchn 02 FF 00 00 00 00 77 77
tx A0 1B CF 02 FF 00 00 00 00 77 77 A4
rx none ok

>> osp send 108 setpwm 00 00 00 00 66 66
tx A4 23 4F 00 00 00 00 66 66 25
rx none ok
```



am^U

OSRAM