Advanced Micro-controllers - ARM

DESD @ Sunbeam Infotech

Conditional branching

- cmp rm, rn → does subtraction and update ALU flags

- bxx label
  - Branching based on condition
  - GT, LT, GE, LE, EQ, NE

ALU flags

- Conditional execution of ARM instructions
  - movxx rd, rs
  - addxx rd, rm, rn

} not supported in Thumb state

- Thumb-2 if-then instruction → max 4 instructions
  - cmp rm, rn
  - ite gt → sets IT bits of xPSR
  - movgt rd, rm
  - movle rd, rn

} execute as per IT bits and modify IT bits.

ite
ittee
ittte
itee

To write execute lines based on condition

```
cmp r1, r2
ite eq
beq label1
bne label2
```

label1:

label2:

Barrel shifter

LSL : Logical Left Shift

Multiplication by a power of 2

LSL r0, #1

ASR: Arithmetic Right Shift

Division by a power of 2, preserving the sign bit

ASR r0, #1

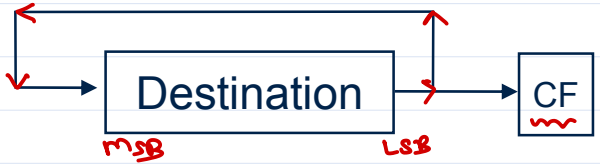
LSR : Logical Shift Right

like (unsigned >> n)

Division by a power of 2

LSR r0, #1

ROR: Rotate Right



Bit rotate with wrap around from LSB to MSB

ROR r0, #1

RRX: Rotate Right Extended

Single bit rotate with wrap around from CF to MSB

RRX r0, #1

shift operation on 2nd operand.

- Inline barrel shifter
  - mov rd, rs, lsl #k
  - mov rd, rs, lsr #k

Load store instructions

- Global variables
  - .section .data
  - num1: .word 10

- ldr ra, =addr rd = &addr

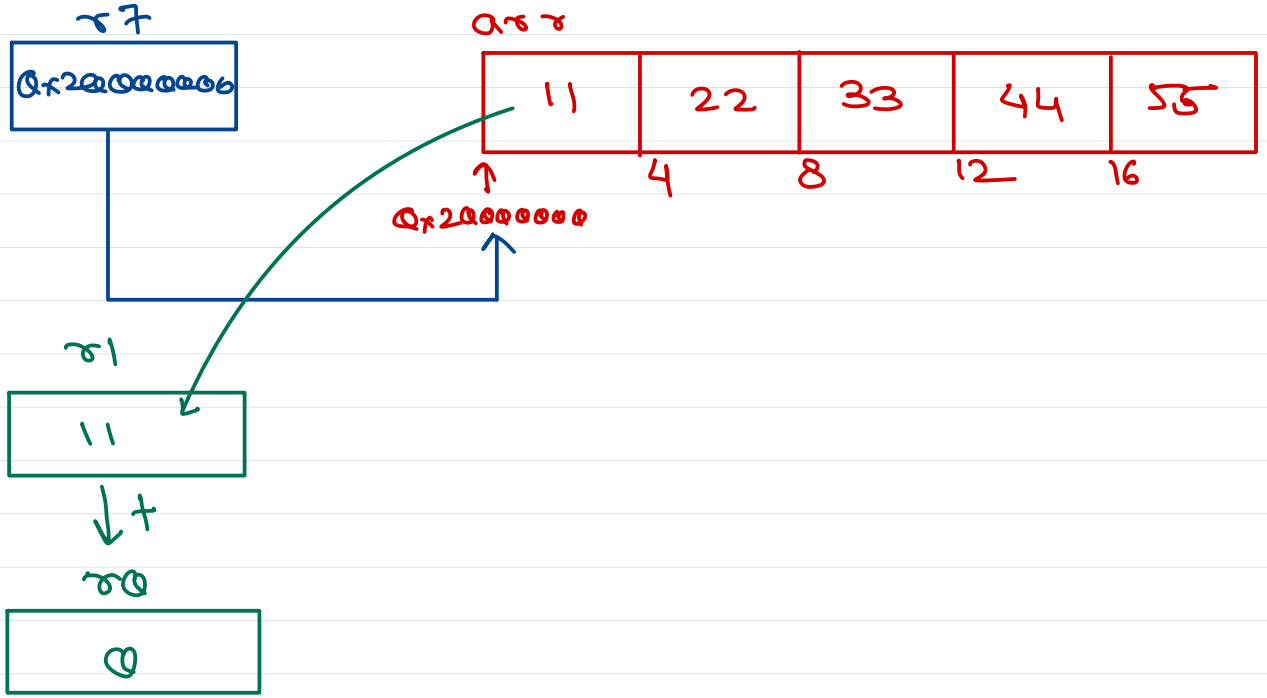
- ldr rd, [ra] rd = *ra

- str rs, [ra] *ra = rs

byte half word word double word

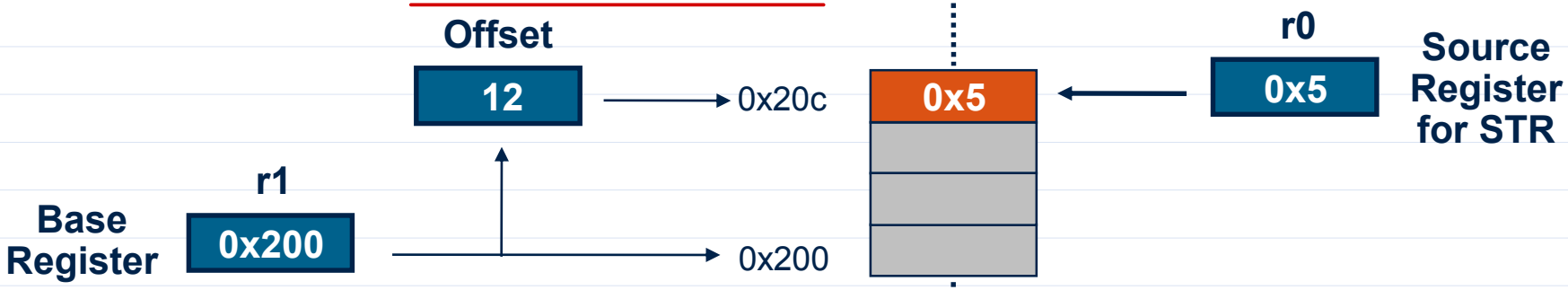
- ldrb, ldrh, ldr, ldrd
  strb strh str strd
- ldrsb, ldrsh

signed byte signed halfword



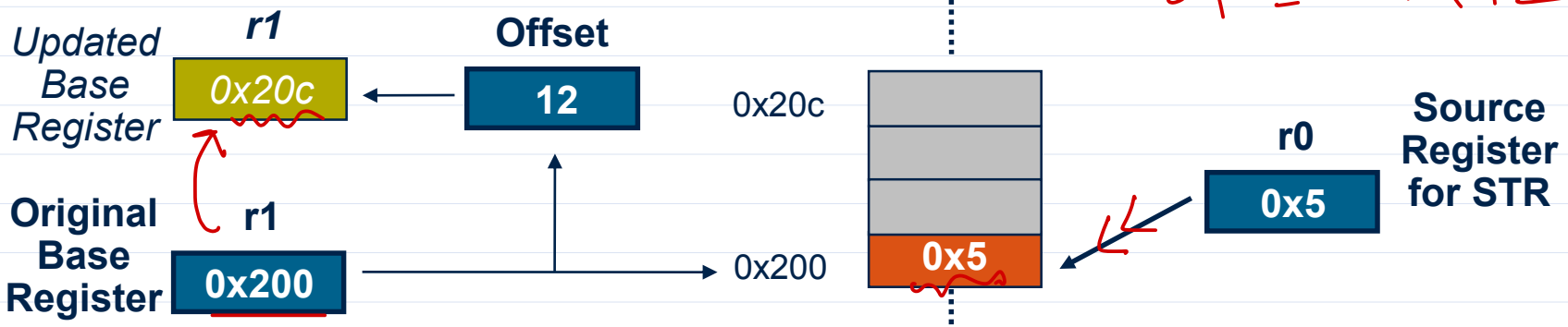
Load/Store Pre/Post-increment

- **Pre-indexed:** `STR r0,[r1,#12]`

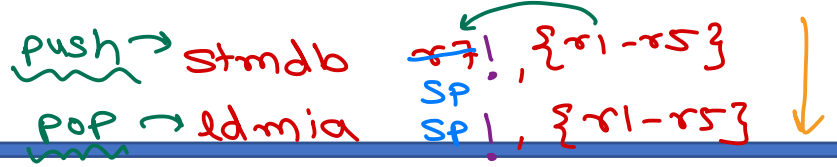


Auto-update form: `STR r0,[r1,#12]!`

- **Post-indexed:** `STR r0,[r1],#12`



Load/Store Multiple



- Syntax:

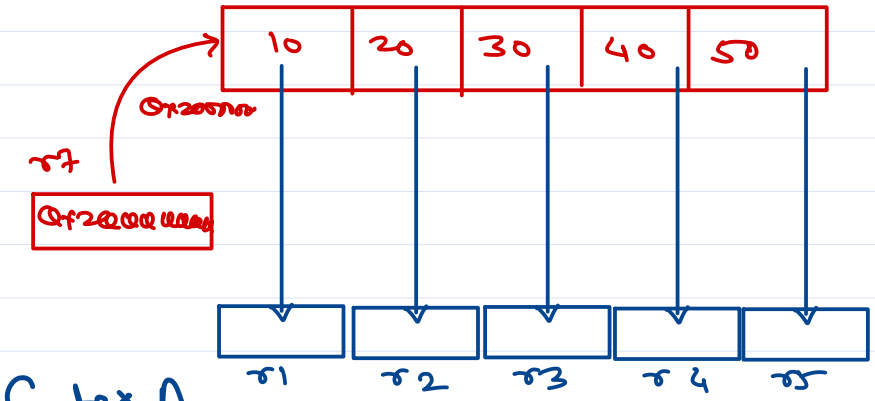
  `<LDM|STM>{<cond>}<addressing_mode> Rb{!}, <register list>`

- 4 addressing modes:

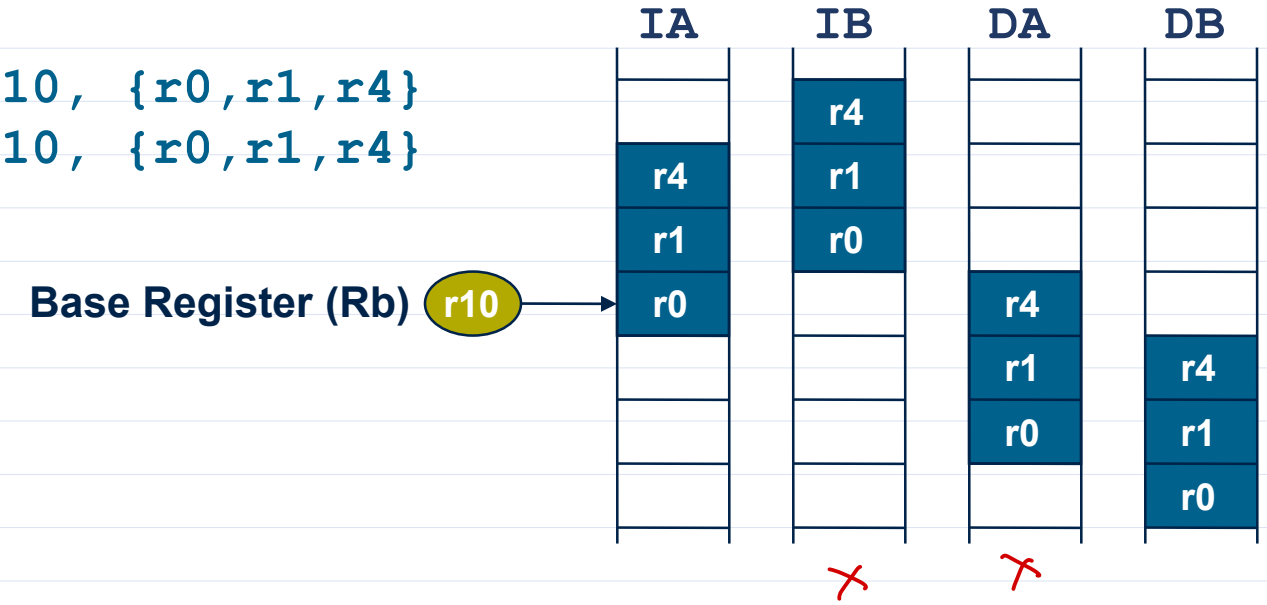
`LDMIA / STMIA`	increment after
~~`LDMIB / STMIB`~~	~~increment before~~
~~`LDMDA / STMDA`~~	~~decrement after~~
`LDMDB / STMDB`	decrement before

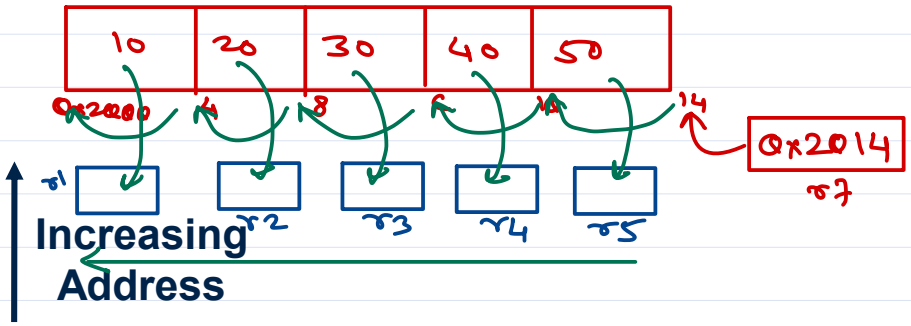
not supported in Cortex-M

all supported in Cortex A.

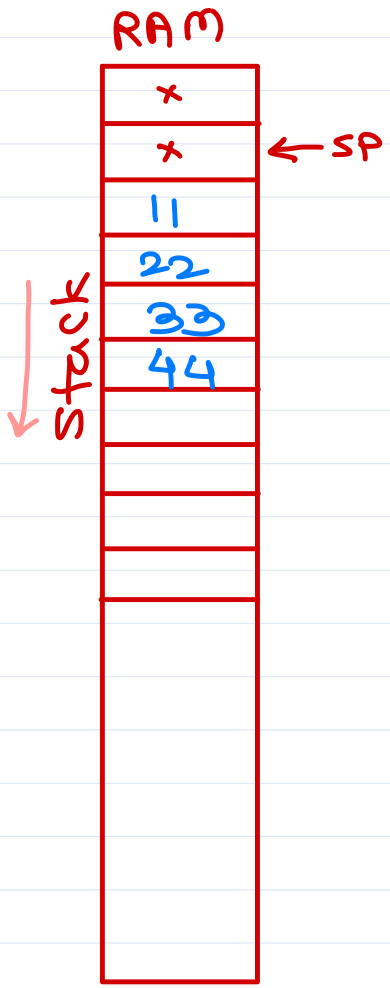


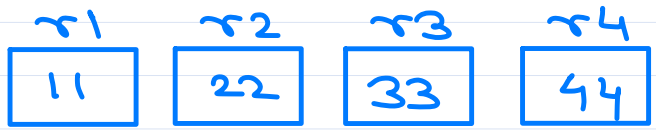

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



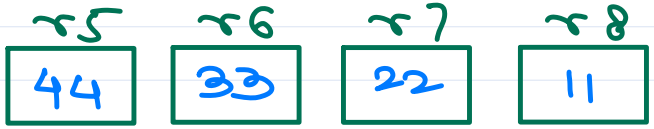


Stack Operations





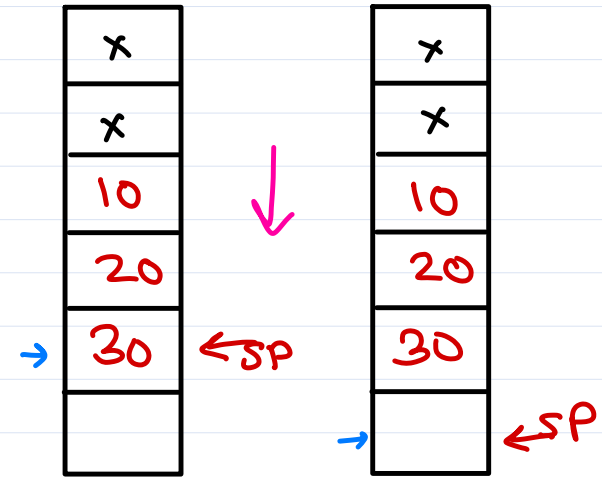
STMDB SP!, {r1-r4}

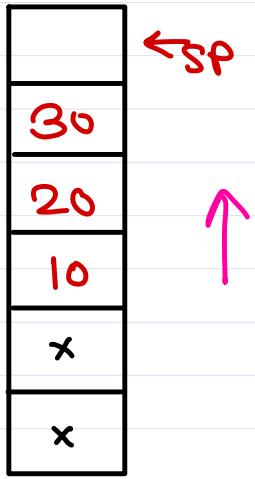


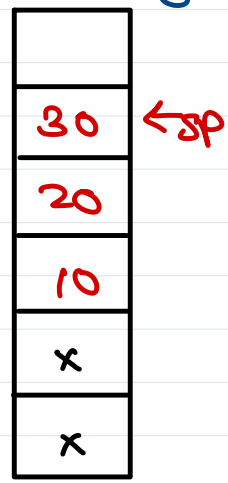
LDMIA SP!, {r5-r8}

Cortex-M

Cortex-A

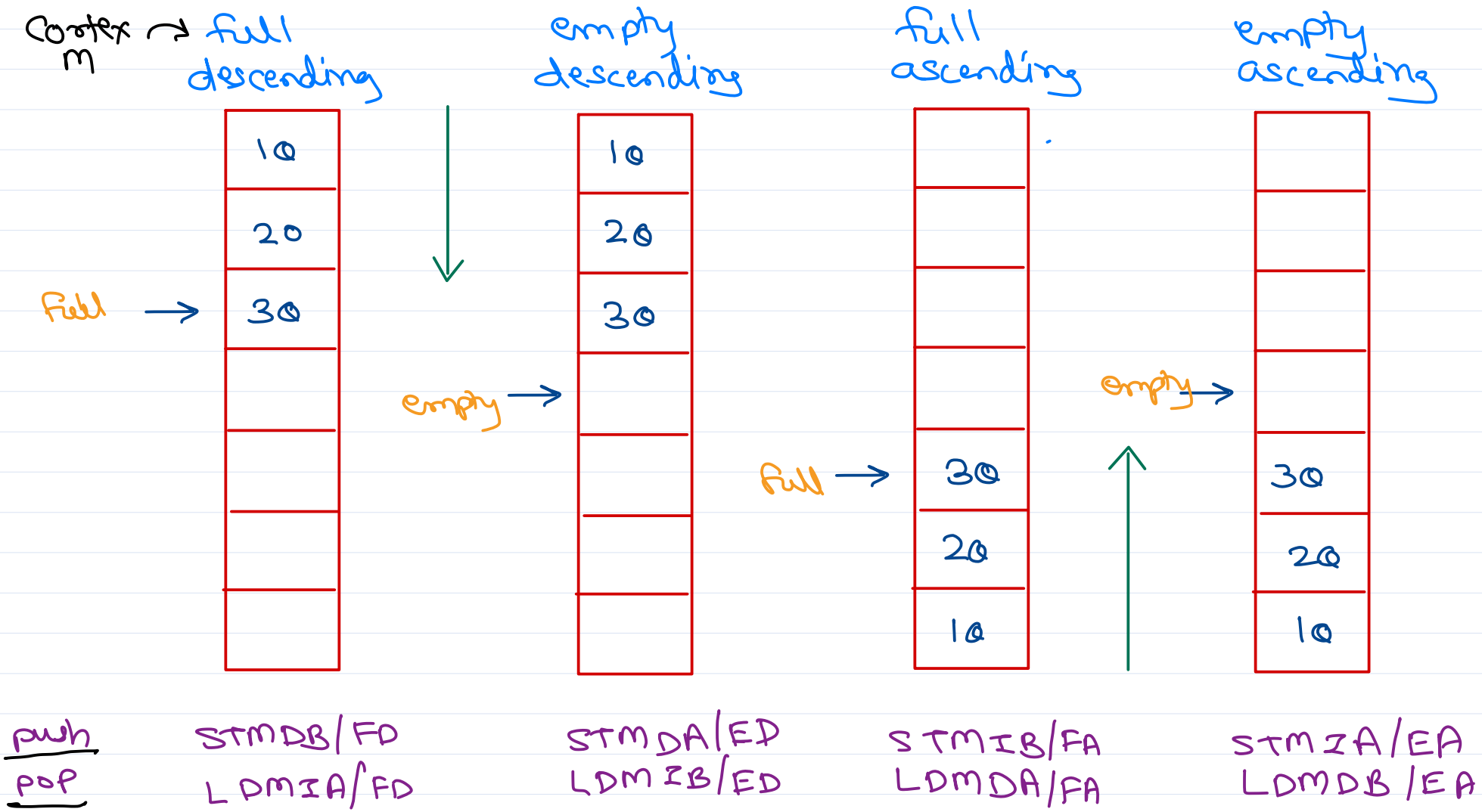




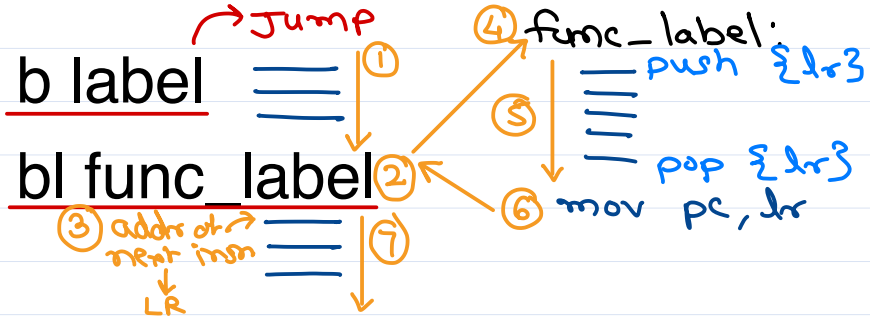


	full descending	empty descending	empty ascending	full ascending
push:	STMDB	STMDA	STMIA	STMIB
pop:	LDMIA	LDMIB	LDMDB	LDMDA
push:	STMFD	STMED	STMEA	STMFA
pop:	LDMFD	LDMED	LDMEA	LDMFA

ARM Stack types



Function call and stack operations



- b label
- bl func_label
- stmfd sp!, {lr}
- push {lr}

} Store lr on stack

- mov pc, lr

- ldmfd sp!, {pc}
- pop {pc}

} Pop return addr into pc from stack

- AAPCS
  - Arguments: r0, r1, r2, r3
  - Return value: r0

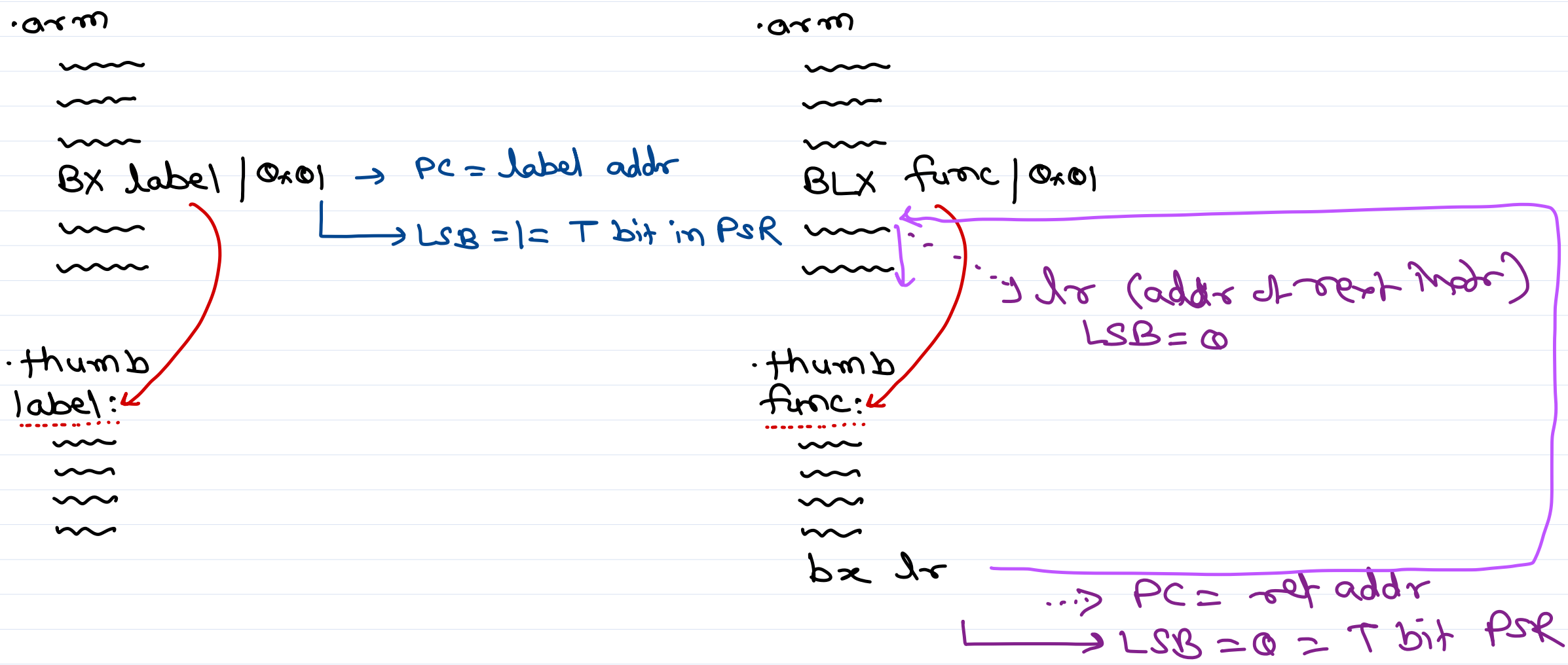
arg1 arg2 arg3 arg4 arg5..... pushed on stack

  - Called saved: r4-r11, lr
  - Caller saved: r0-r3, r12

(4 bytes) ldr or str → sp
(2 bytes) ldrh or strh
(1 byte) ldrb or strb
(8 bytes) ldrd or strd

expect addr to be dword aligned (multiple of 8)

ARM Thumb Interworking



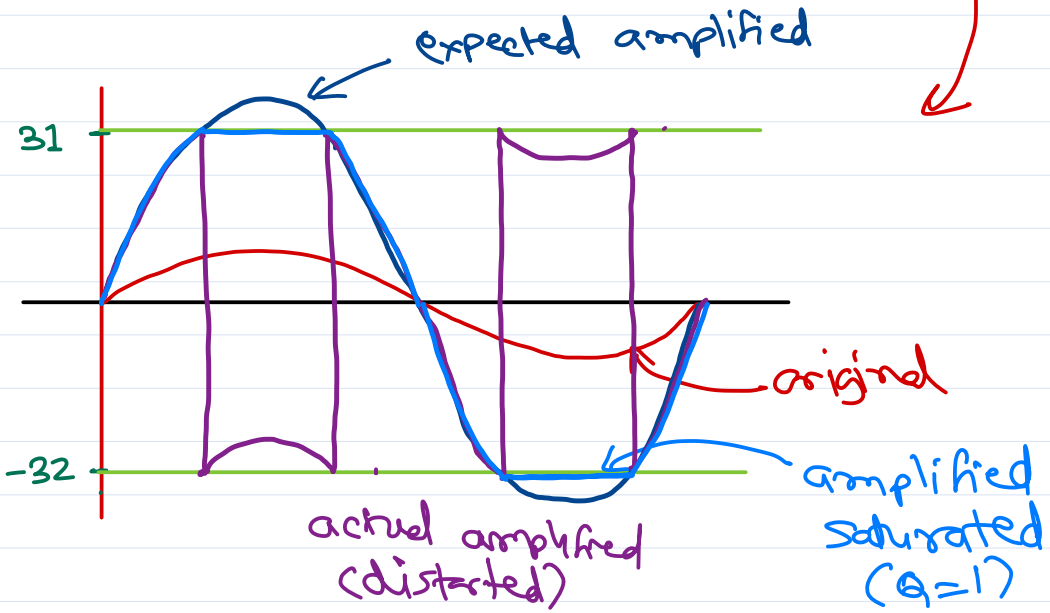
DSP instructions

- Saturated Math
  - mov r0, #10
  - usat r1, #5, r0, lsl #1
  - usat r1, #5, r0, lsl #2

@ r1 = MIN(r0 * 2, 31) --> 20 (q=0)

@ r1 = MIN(r0 * 4, 31) --> 31 (q=1)

- SIMD instructions
  - ldr r1, =0x11223344
  - ldr r2, =0x44332211
  - qadd8 r0, r1, r2



Miscellaneous instructions

- rev instruction
  - ldr r0, =0x11223344
  - rev r1, r0 @ r1 -- 0x44332211
- sign extend
  - ldr r0, =0x55AA8765
  - sxtb r1, r0 @ last byte of r0 -- 0110 0101 @ new value of r1 will be -- 0x00000065
  - sxth r2, r0 @ last 2 bytes of r0 -- 1000 0111 0110 0101 @ new value of r2 will be -- 0xffff8765
  - uxth r3, r0 @ new value of r3 will be -- 0x00008765
- bit-field extrac
  - ldr r6, =0x11223344 @ assume value of ADGDR is 0x11223344
  - ubfx r0, r6, #4, #12 @ new val of r6 will be = 0x0334
- clear/insert bits
  - ldr r1, =0x11223344
  - bfc r1, #8, #16 @ r1 will be 0x11000044
  - mov r0, 0x12
  - bfi r1, r0, #8, #8 @ r1 will be 0x11001244

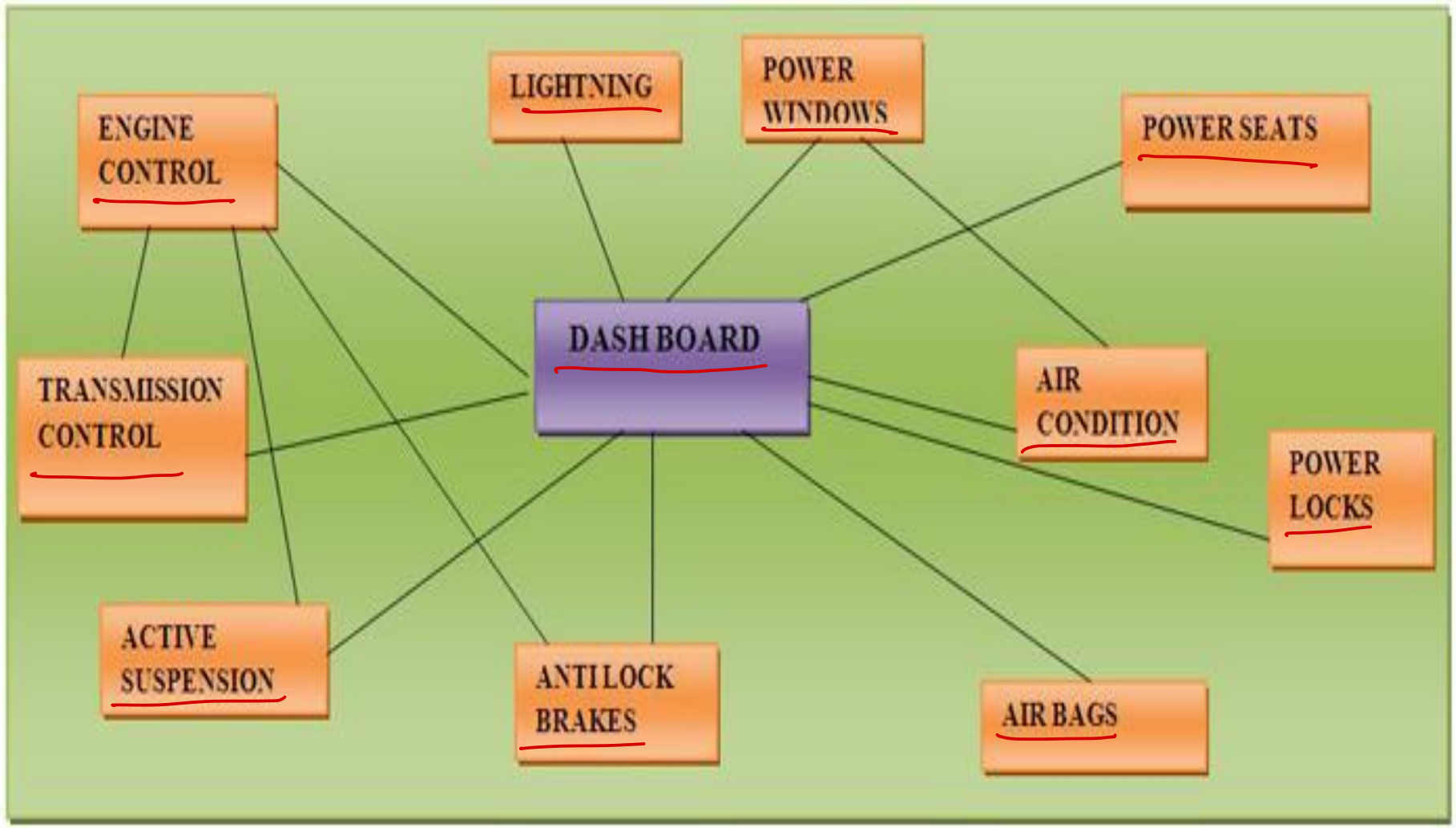
CAN Protocol

Controller Area Network

Introduction

- CAN protocol is method of communication between various electronic devices. It defines set of rules for communication in a network of devices.
- Original idea initiated Robert Bosch in 1983. However first release of CAN protocol is done in 1986.
- Protocol is implemented in hardware and software to communicate between different controllers present in the automobiles.
- Nowadays this protocol is used in various industries including Healthcare (ICUs & Operation Rooms), Entertainment (light control, door control in studios, gambling machines), Science (high energy experiments, astronomical telescopes).

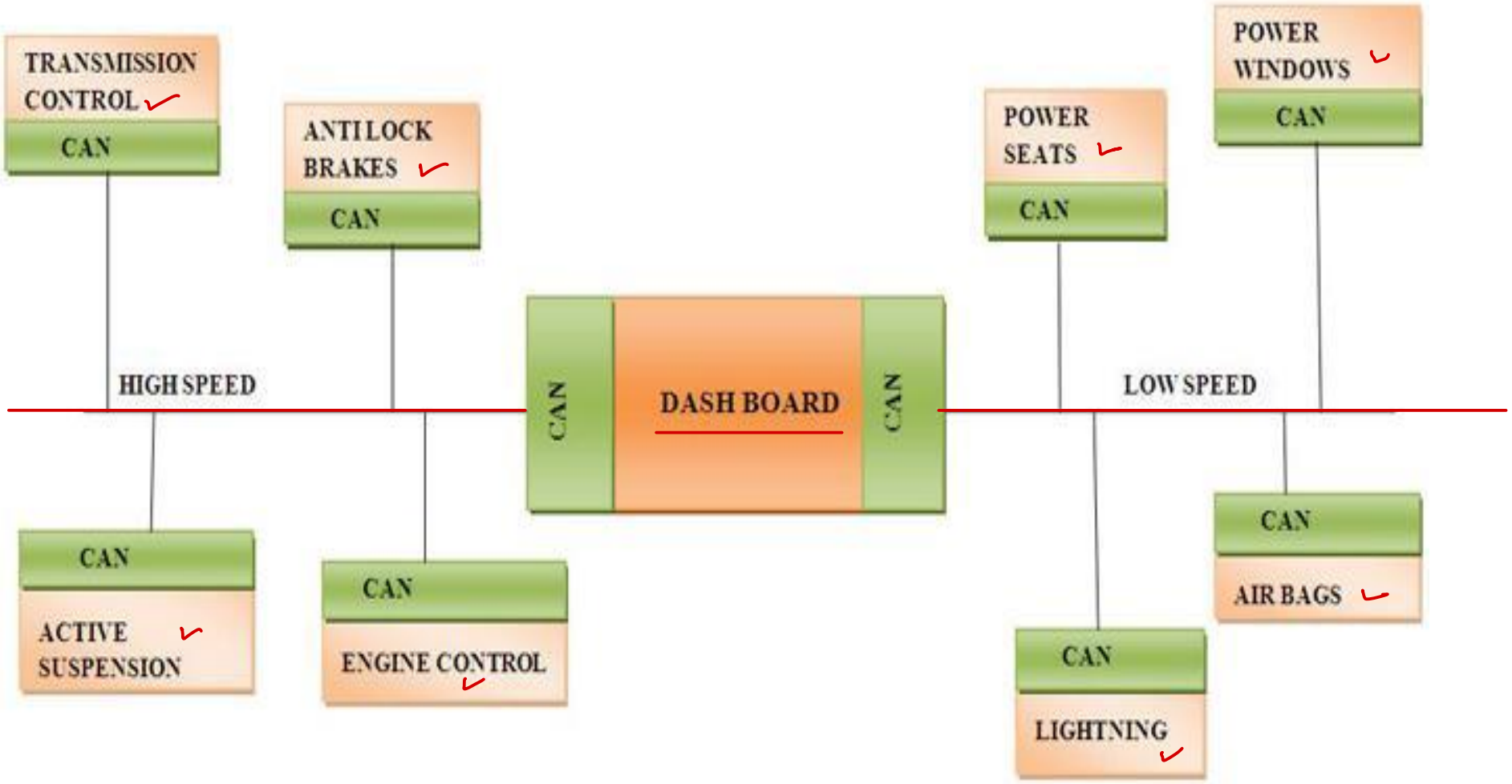
Automobiles – Prior invention of CAN



Drawbacks & Limitations of Wired System

- Number of wires in various subsystem makes the system complicated and difficult to maintain.
- Passing real time information among subsystems was tedious implementation (serial protocols used).
- Asynchronous transmitter/receiver do not support multi-domain communication e.g. communication between air-conditioning system and door/window system.
- Multiple domains in automobiles includes power generation (engine), chassis (driving mechanism), body (climate control/wipers), telemetric (entertainment units) and passive safety (air bags, etc).

Automobile System – with CAN protocol

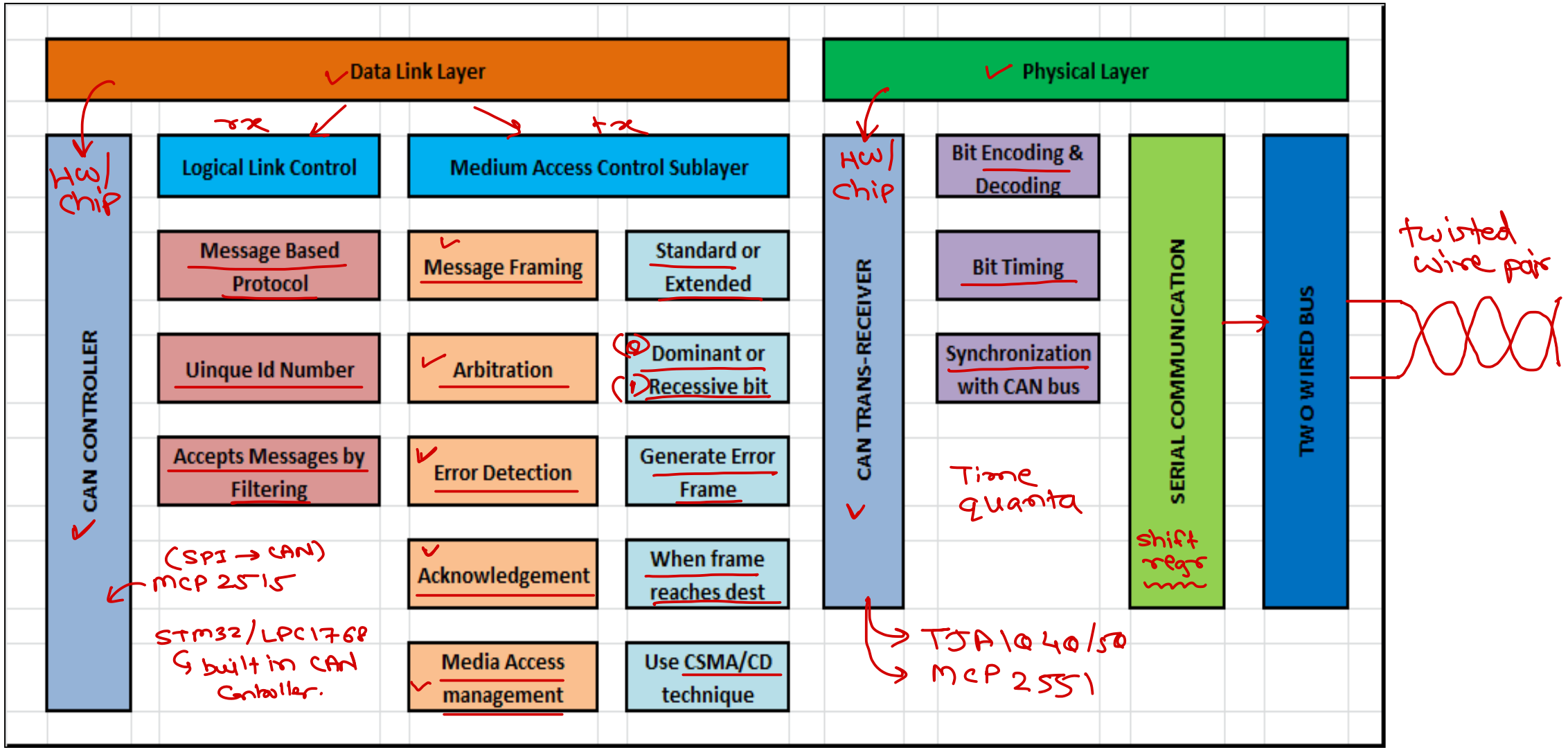


CAN Architecture

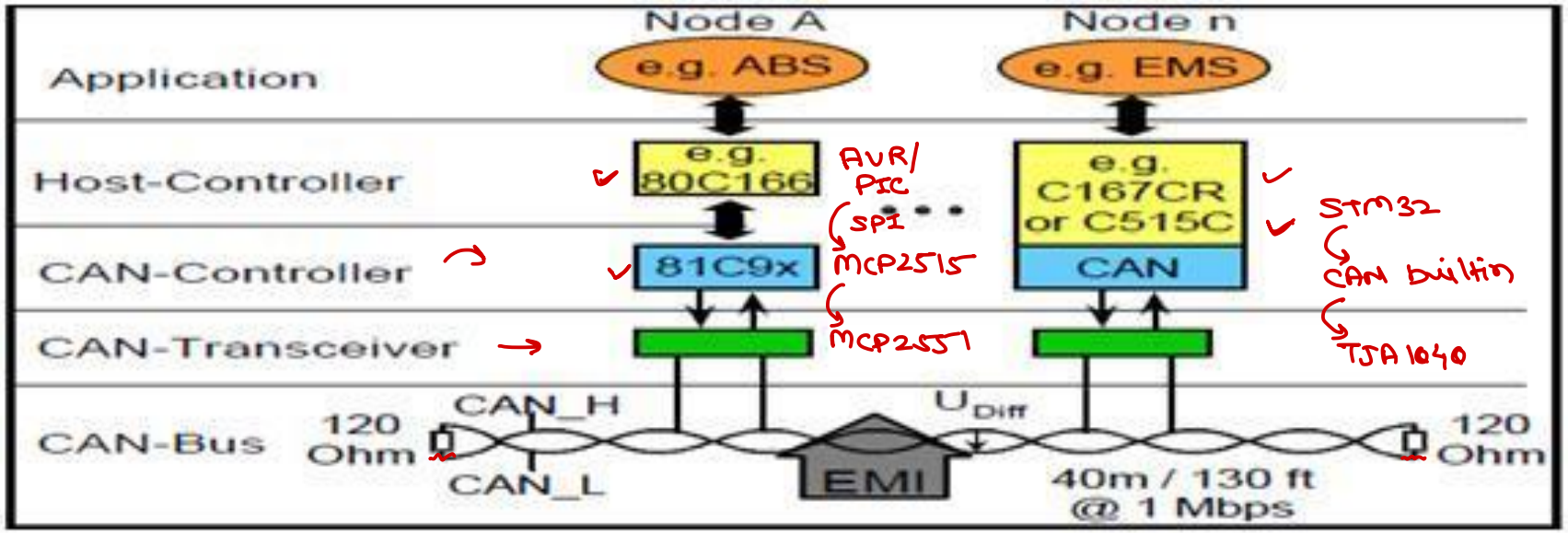
- CAN protocol is implemented with OSI reference model.
- It implements two layers of OSI model and rest are left for implementation specific to the requirement.
- Data Link Layer
  - Logical link control
    - Allows filtering of messages based on UID.
  - Medium access control
    - Prepare message frame and handle arbitration.
- Physical Layer
  - Send bits to the CAN two wire bus as per timing requirements.

- application layer
- presentation layer
- session layer
- transport layer
- network layer
- data link layer
- physical layer

CAN Architecture

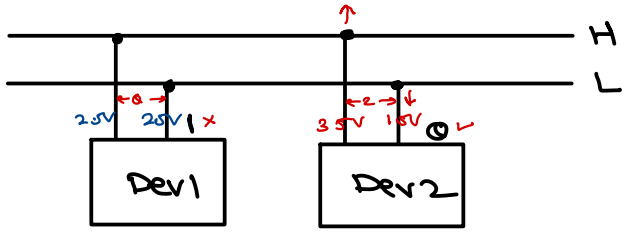


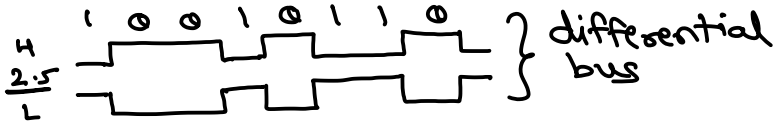
CAN Node

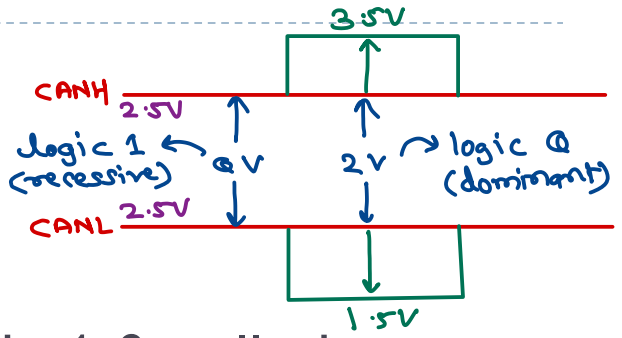


- Each electronic device is called as Node.
- Host-controller is MCU responsible for functioning of node.
- CAN controller converts messages of node as per CAN protocol. Can be a separate chip or embedded in MCU.
- Trans-receiver is to transmit bits on CAN bus.

CAN Bus





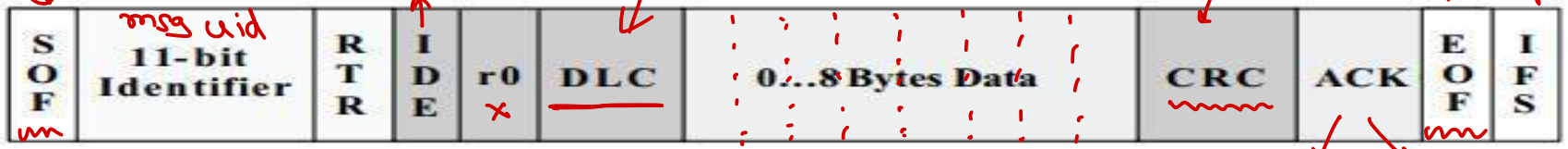


- CAN bus is a two twisted wire bus i.e. CANH & CANL.
  - The passive voltage of each line is 2.5 V.
  - The active voltages are 3.5 V and 1.5 V.
  - When both lines are 2.5 V, difference is 0 V. It represent logic 1 & called as “recessive bit”.
  - When both lines are pulled to 3.5 V and 1.5V respectively, then difference is 2 V. It represent logic 0 & called as “dominant bit”.
  - Note that dominant bit can always overwrite recessive bit.
  - CAN bus is a linear bus terminated with 120 Ω. Also input impedance of each node is 120 Ω.
- CAN bus is not a master slave bus i.e. Any node can write the data on the bus in certain format (frame) provided bus is available.

CAN Frame

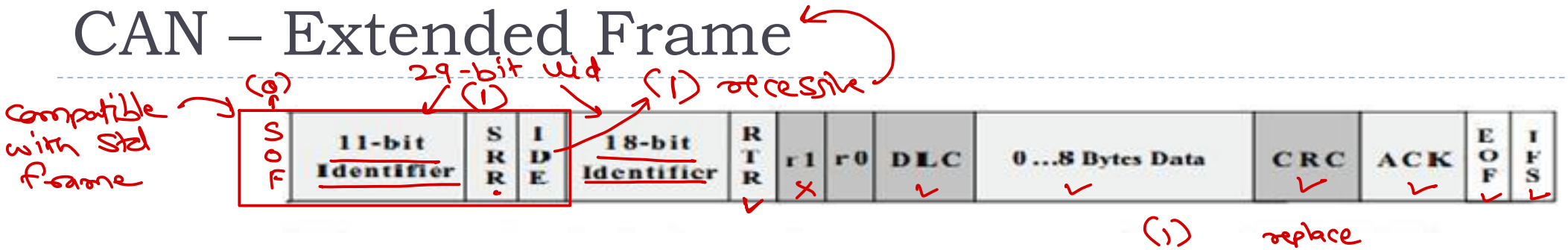
- CAN is a message based protocol (not address based).
- Message contains a pre-defined unique id (rather than addresses).
- Messages are accepted or rejected by any node based on this UID. If multiple nodes send messages at same time, node with highest priority gets bus access.
- CAN message is made up of 10 bytes.
- Each message is coded into meaningful sequence of bits/bytes called as **frame**.
- Framing is done by Medium Access Layer.
- There are two types of frames:
  - Standard CAN Frame
  - Extended CAN Frame

CAN – Standard Frame



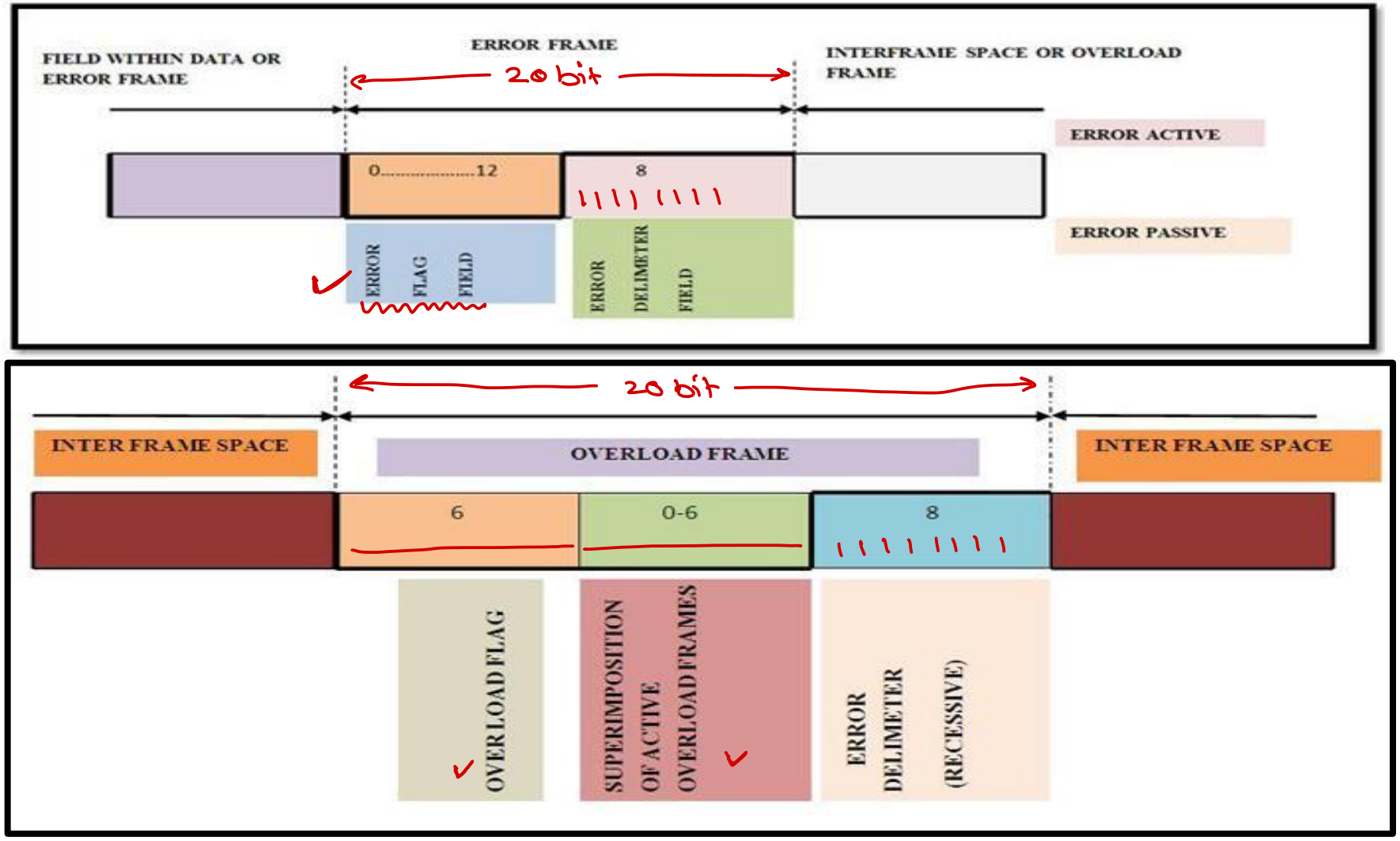
- SOF: Start of Frame – Dominant bit – For Sync.
- UID: Type of message and Arbitration.
- RTR: Type of frame i.e. Data Frame [Dominant] or Remote Transmission Request (RTR) Frame [Recessive].
  - RTR frame don't have data, instead request other node to send data.
- IDE: UID Extension. Standard (dominant) or Extended (recessive) frame.
- R0: Reserved for future use.
- DLC: 4-bit data length code [0 to 8 bytes – data length]
- DATA: 0 to 8 bytes
- CRC:15 bits CRC + 1 bit delimiter (recessive)
- ACK: Transmitter sent recessive bit, Rcvr overwrite with dominant. + 1 bit delimiter (recessive)
- EOF: 7-bits (recessive) to indicate End of Frame.
- IFS: 3 recessive bits - Intermission bits – Separation between two frames.

CAN – Extended Frame



- SRR: Substitute Remote Request : Recessive bit [~~like~~ RTR]
- 11-bit Id + 18 bit Id = 29 bit Id – Extended Message Id.
- R1: Additional reserved bit.
- Types of Frames:
  - Data Frame
  - Remote Frame
  - Error Frame: When error is detected, transmission aborts and send this frame.
  - Overload Frame: Like error frame, sent by node when busy in internal processing.

CAN – Error and Overload Frames

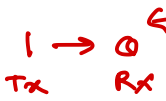


CAN Error Detection & Handling

- There are five methods of error detection.
  - Message Level Error Detection.
    - CRC check
    - ACK slots
    - Form error
  - Bit Level Error Detection.
    - Stuff error
    - Bit error
- If node detects an error, following steps occurs:
  - Transmits error flag.
  - Destroys transmitted frame.
  - Transmitting node resends the frame.

CAN – Error Detection

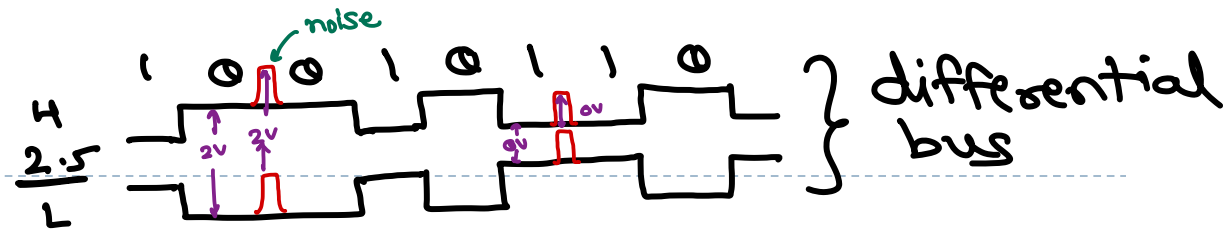
- CRC check:
  - Calculated and sent by transmitter node.
  - Receiver node recalculate CRC and if differs, raise error.
- ACK slots:
  - Transmitter send recessive bit & Receiver overwrite dominant
  - If none of the node overwrite dominant bit, error is raised.
- FORM (FORMAT) error:
  - EOF, IFS, ACK delim bits are always recessive.
  - If dominant bit is found, error is raised.
- BIT error:
  - Transmitter always monitor sent bit.
  - If sent bit is not validated error is generated, except in case of arbitration and acknowledgment bit.



CAN – Error Detection

- Bit Stuff Error:
  - CAN bus is never IDLE as it follows NRZ method (non-returning to zero i.e. 0 & 1 is represented as non-zero values - differential).
  - For sake of synchronization one bit of opposite polarity is added after consecutive 5 bits of same polarity, called as bit-stuffing.
  - Stuffed data frames are de-stuffed by data link layer of receiver.
  - If error is found in stuffing, error is raised.
  - In CAN, 6 consecutive recessive/dominant bits represent error bits.
  - All fields in the frame are stuffed with the exception of the CRC delimiter, ACK field and end of frame which are a fixed size.

CAN protocol – Advantages



- Low cost: Only two wire serial bus.
- Reliable: Error detection & handling. Immune to noise.
- Flexibility: Nodes can be easily added or deleted.
- High speed: Support data rate of 1 Mbits/sec @ 40m bus.
- Multi-master bus: Any node can access bus.
- Fault confinement: Faulty nodes do not disturb commn.
- Broadcast capability: One to One/Many/All commn.
- Standardization: ISO standardized.
  - ISO-DIS 11898 : High speed communication
  - ISO-DIS 11519-2 : Low speed communication

Thank You!

Nilesh Ghule

<nilesh@sunbeaminfo.com>