
CAN202C Ex2
CAN ID =x0150
CAN202C Workshop
MICROCHIP
CAN BUS Analyzer
USB STAT
USB BUSY
USB POWER
GND
VBAT
TRIGGER
STATUS
ACTIVE

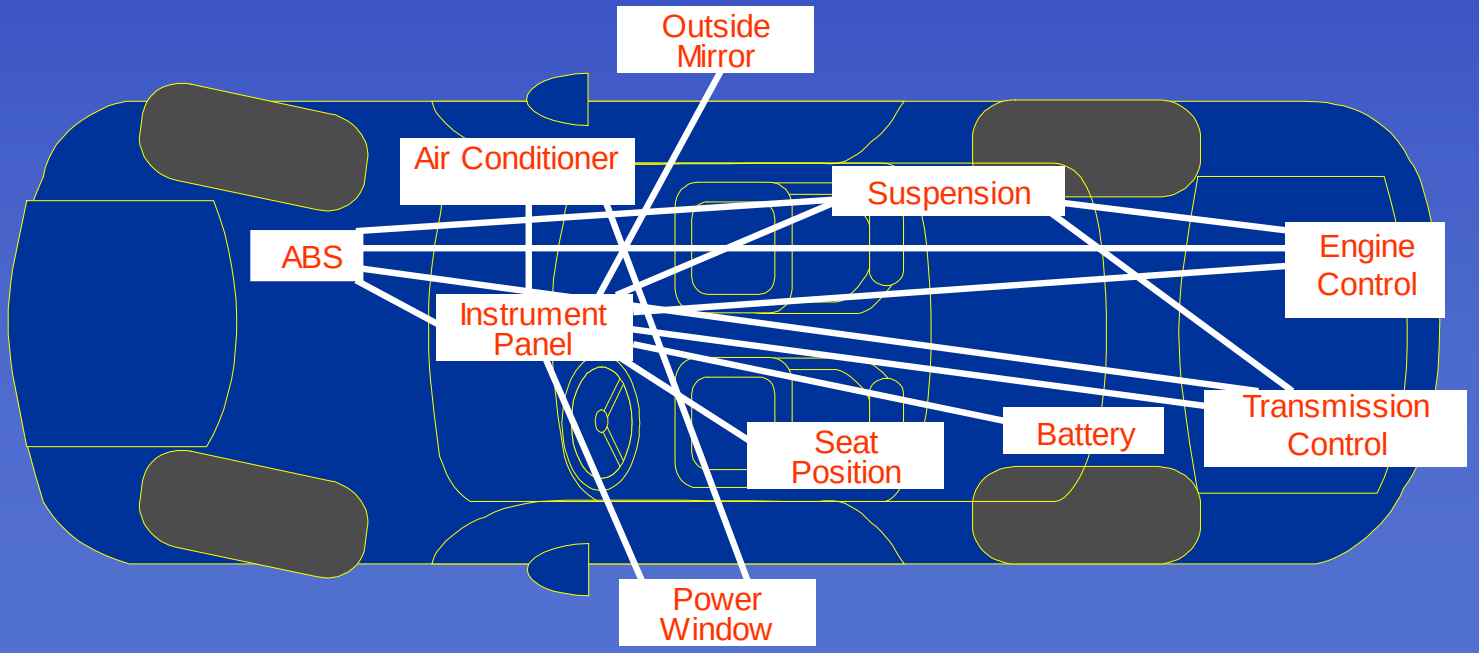
CAN Protocol

使用 **CAN bus** 來實現
高速而可靠的通信

What's CAN Bus

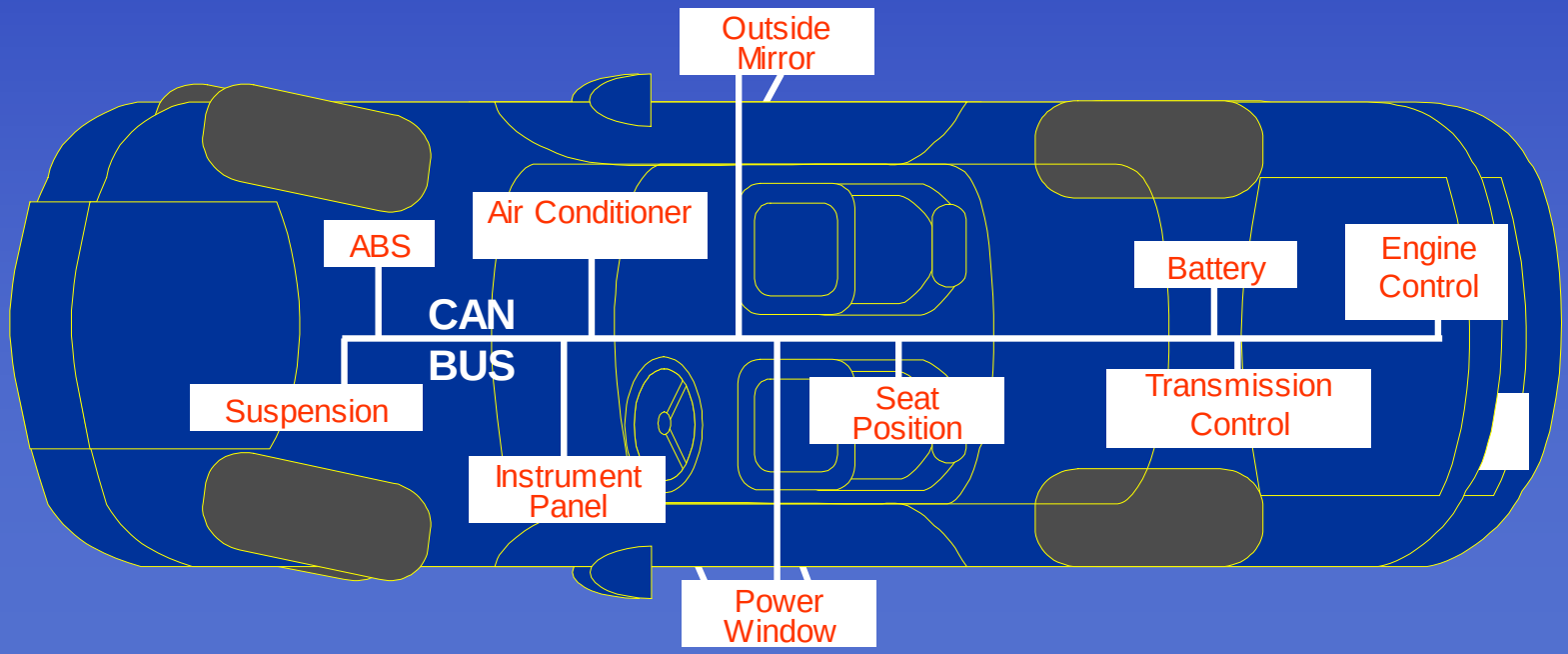
CAN (Controller Area Network) 通信協定於 **80** 年代由 **Bosch** 首先發展, 為的是因應使用於新型汽車上不斷增加的電子裝置. 這些裝置使汽車增加許多功能與附加價值, 也增加控制系統的複雜度

在 **CAN Bus** 尚未被使用之前, 控制系統與各感測器之間大多透過點對點的電纜連接以完成控制及資料交換



CAN Bus 的優點

在 CAN Bus 尚未被使用之前,控制系統與各感測器之間大多透過點對點的電纜連接以完成控制及資料交換
使用 CAN bus, 透過簡單的串列界面即可完成對整個控制系統的連接及控制



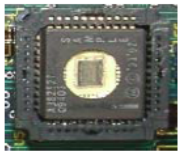
CAN 規範的發展沿革

84 **Bosch** start development on CAN

85

86 CAN **patent** filed 

87 CAN published at **SAE congress Detroit**

88 First **CAN chips** from Intel and Philips 

89

90 CAN introduced first in **weaving machines**

91 First Mercedes-Benz **S-class** with CAN 

92 Foundation of CAN in **Automation**

93 Standardization of CAN in **ISO 11898**

94

95 **CANopen** protocol published by CiA

96

97

99 

00 Introduction of **TTCAN**

01

02

03 **Specification of several ISO 11898-x:**

04 data link layer

05 high-speed physical layer

06 fault-tolerant physical layer

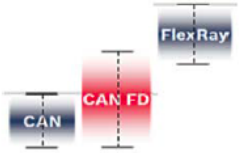
07 TTCAN

08 low-power mode

09 selective wake-up

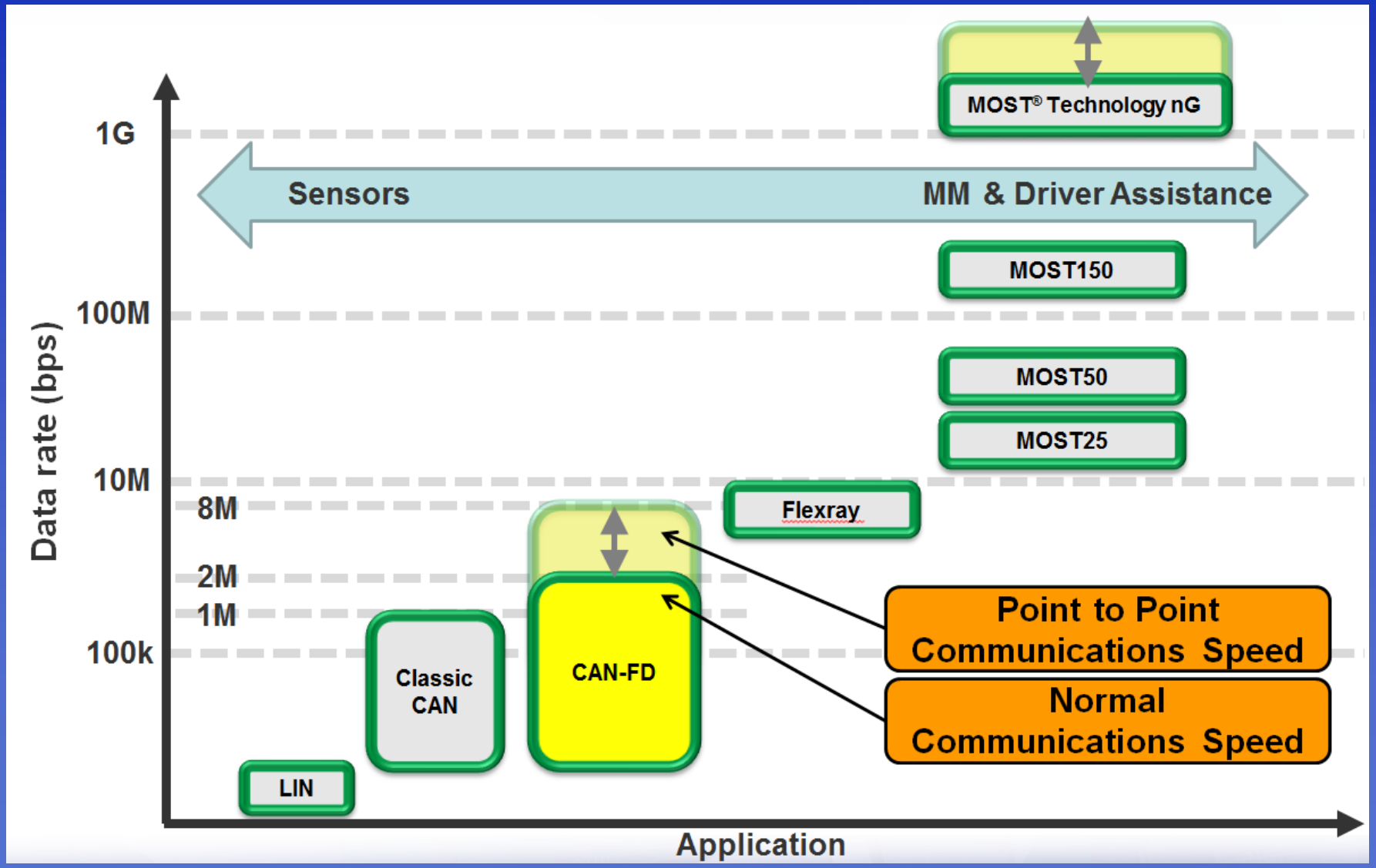
10

11 Invention of **CAN FD** (ISO 11898-7)

12 

Source: CiA

幾種常見的車用 Bus 之效能比較



現階段汽車應用的挑戰

- **Slow End of Line (EOL) Programming**
  - Increase of Flash size
- **Limited Operating Bandwidth**
  - Continued growth of features/messages
  - Splitting of CAN buses in vehicle to support bandwidth
- **Fragmented CAN Messages**
  - Need more than 8 data bytes (authentication)
- **Functional Safety**
  - Improved error detection

CAN FD 如何被用來解決這些問題

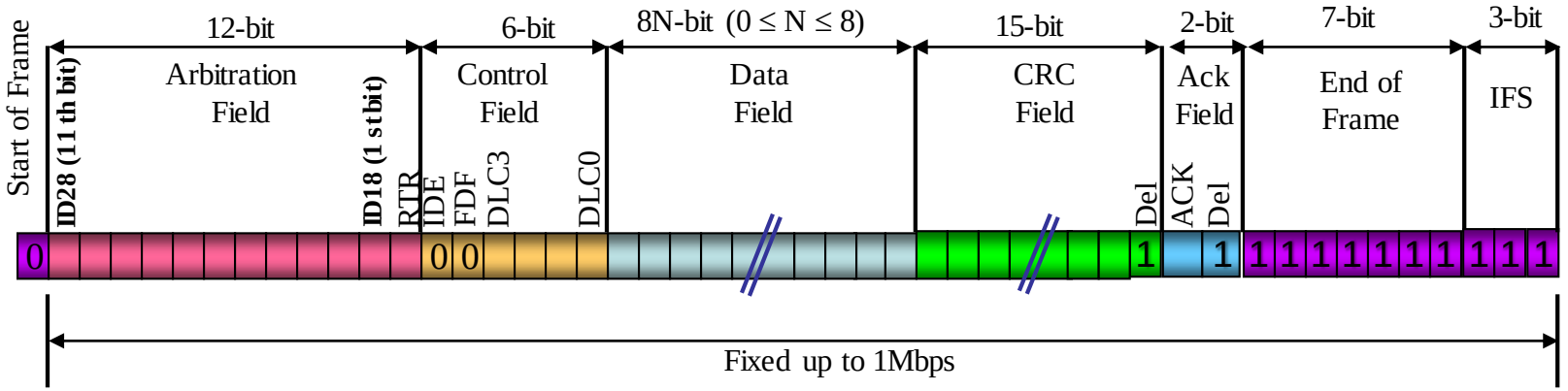
- **Increases typical system data rate**
  - From ~500kb/s to 2Mb/s (normal operation) & up to 5Mb/s (diagnostics or programming)
- **Increase payload from 8 data bytes in the CAN message to 64 data bytes**
- **Benefits of this new protocol**
  - Significantly decreases EOL programming
  - Frees up bus bandwidth allowing more ECUs
  - Reduces post processing for data transmitted in multiple CAN messages

與 CAN FD 相關的 ISO Standard

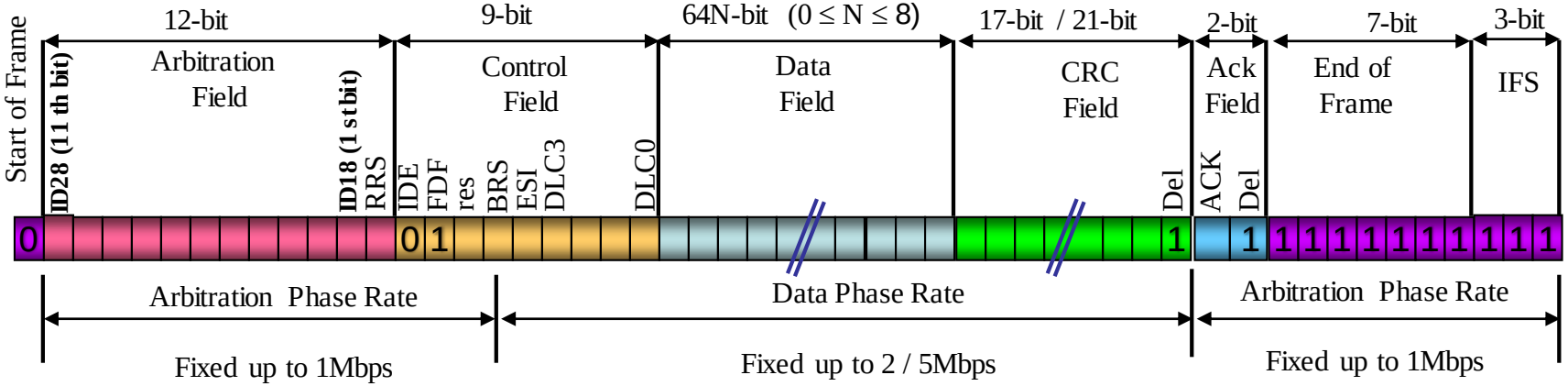
- **CAN FD → ISO11898-1:2015**
  - It describes classical CAN and CAN FD
  - Describes CAN data link layer (how digital information is interchanged between CAN nodes)

- **ISO11898-2:2016 describes the CAN FD Physical Layer requirements**

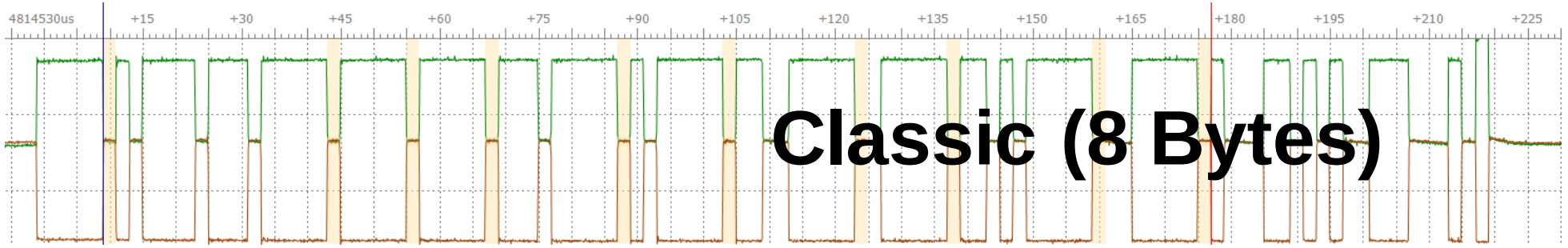
CAN classic standard format



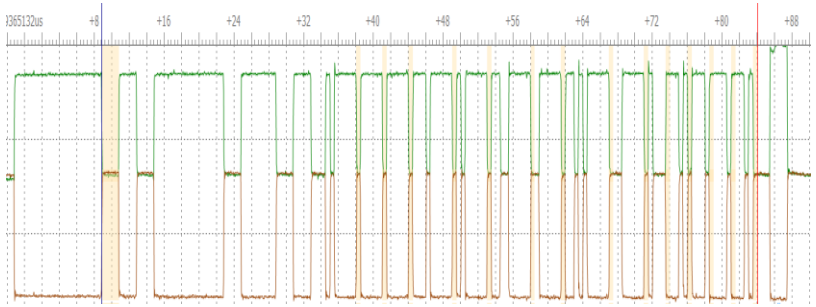
CAN FD standard format



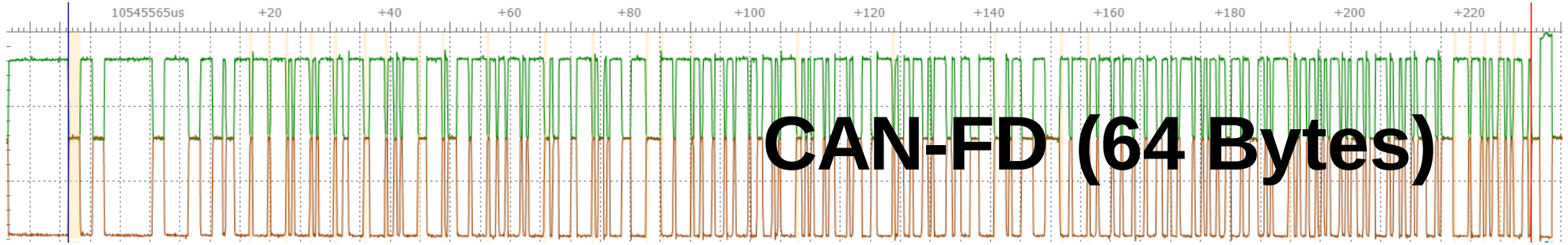
CAN (Classic) vs. CAN FD Frames



Classic (8 Bytes)



CAN-FD (8 Bytes)



CAN-FD (64 Bytes)

CAN Network Solutions
Bosch Specifications

- ***Bosch Spec 2.0B (Active)***
  - ✓ CAN 通信協定的最新版本
  - ✓ 可包含 29 bit 的 Identifier (Extended Frame)
  - ✓ 完全地相容於較早期的版本 - Bosch spec 2.0A

  - ✓ 所有 Microchip 的 CAN devices 都與 CAN 2.0B 相容
  - ✓ Microchip 的 ATSAM MCU 有對 CAN-FD 的 Support

CAN Network Solutions
Bosch Specifications

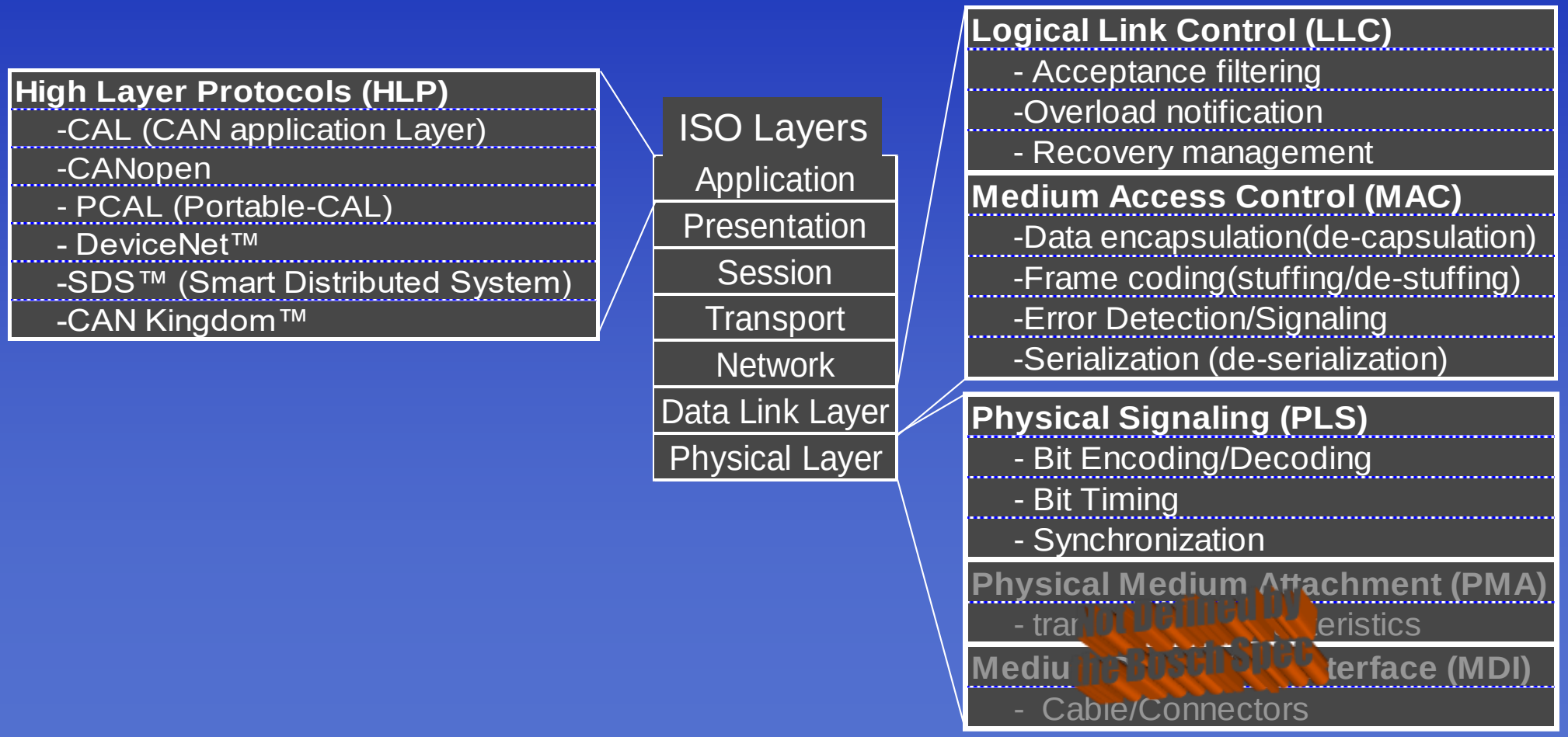
- ***Bosch Rev 2.0A***
  - **Bosch 2.0A 的規格中共有 2 種不同的版本**
    - ***Rev2.0A*** **– 此版本的通信協定只能傳送與接收 “standard frame message” 的資料 (11-bit Identifier) . 假設接收到一個 extended frame message (29-bit Identifier) 則將產生 “error flag “**
    - ***Rev2.0B Passive*** **– 此版本的通信協定依然只能傳送及接收屬於 2.0A 規範的 message (Standard Frame)**
      - 當接收到 “Extended Frame Message”，它將會回應 acknowledge 但是會忽略此一 message !
      - 至少錯誤不再發生而且不會因為錯誤而干擾 CAN Bus

CAN Bus 概要總覽

- CAN 是個先進的串列通信協定, 能有效率地支援分散式控制系統 ，它由以下兩個國際組織制定標準
  - ✓ International Standards Organization (ISO)
  - ✓ Society of Automotive Engineers (SAE)
- CAN 被廣泛的應用於汽車及工業控制上, 其他的應用尚包含船舶, 火車, 大樓自動化以及醫療儀器
- 如同大多數的網路系統 (Ethernet, USB, CAN, etc.), 分層的系統架構被使用來達成以下功能
  - ✓ 使來自於不同製造商的產品能互相溝通
  - ✓ 將相近的功能集合在一起
  - ✓ 做為發展網路協定的骨架
- ISO/OSI Seven Layer 網路參考模型定義了這種有系統的組織架構

BASIC NETWORK OVERVIEW

ISO/OSI Network Layering Reference Model for CAN Bus



CAN Network Solutions
在汽車上的應用

- Driver Seat Control Unit
- Oil Condition and Level Sensor
- Diagnostics and Service Port
- Steering Position Sensor
- Engine Sensor Module
  - ***Temperature Sensor***
  - ***Air Intake Position Sensor***
  - ***Pressure/Vacuum Sensors***
- Airbag Controller
- Environmental Controls
- Engine Control Applications
- Transmission Control
- Anti-Lock Brakes
- Suspension Control
- Dashboard/Instrumentation Control
- Power windows/Moon roof
- Power Mirrors

CAN Network Solutions
在非汽車相關產品的應用

- Controller for Swimming Pool
- Industrial Lighting Controller
- Building Networks
  - Security System
  - Control Systems
  - Elevator Position Sensor
  - Elevator Control Panel
  - Lighting control
  - Emergency Lighting/Warning Systems
- Communication Protocol for industrial Copier
- RAID Storage Systems
- Hospital Bed Controller
- Intercept Seeker for Laser Targeting System
- Battery Charger on Forklift
- GPS navigation system for aviation
- Avionics Control for Model Rockets
- Secure Medicine Dispensing Machine
- Network system for Vending Machines

CAN Bus 的相關規範

- ISO11898-1：定義了 CAN Bus 的 Data Link & Physical Layers
  - ✓ 但不包含 PMA & MDI
- 為了要減輕彼此溝通時將遭遇的問題, ISO 以及 SAE 已經依照 CAN 標準的需求, 制定完成以下包含 PMA (Physical Medium Attachment) 和 MDI (Medium Dependent Interface) 等 ISO 11898-1 未定義的標準
  - ✓ ISO11898-2 是個適用於高速通信的標準規範 (up to 1Mbps)
  - ✓ ISO11898-3 是個適用於較低速通信的標準規範 (up to 125kbps)
  - ✓ SAE J1939 則被制定於使用在卡車及巴士時的應用

ISO 11898-1 為 CAN Bus 所作之定義

Data Link Layer

Logical Link Control (LLC)

- Acceptance Filtering
- Overload Notification
- Recover Management

Medium Access Control (MAC)

- Data Encapsulation/Decapsulation
- Frame Coding (stuffing/destuffing)
- Error Detection/Signaling

Physical Layer

Physical Signaling (PLS)

- Bit Encoding/Decoding
- Bit Timing/Synchronization

ISO 11898-1 對實體層未定義的部分 (尤其他的附加規範來定義)

Physical Layer

Physical Medium Attachment (PMA)

- Driver/Receiver Characteristics

Medium Dependent Interface (MDI)

- Connectors

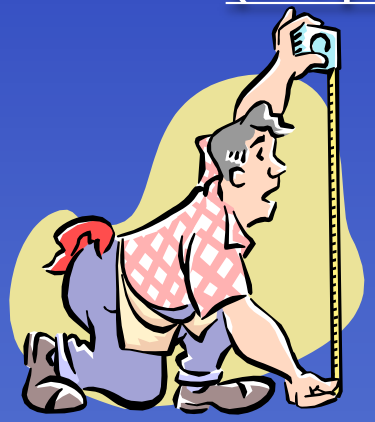
CAN High Speed

- ISO11898-2:2016 specification
- Supports up to 1Mbps bus speed

CAN / Network Solutions
CAN Bus 的能耐

- 資料長度
- CAN 在單次傳送中可最多傳出 8 Byte 的資料
- 傳送速率 **vs.** 傳送距離

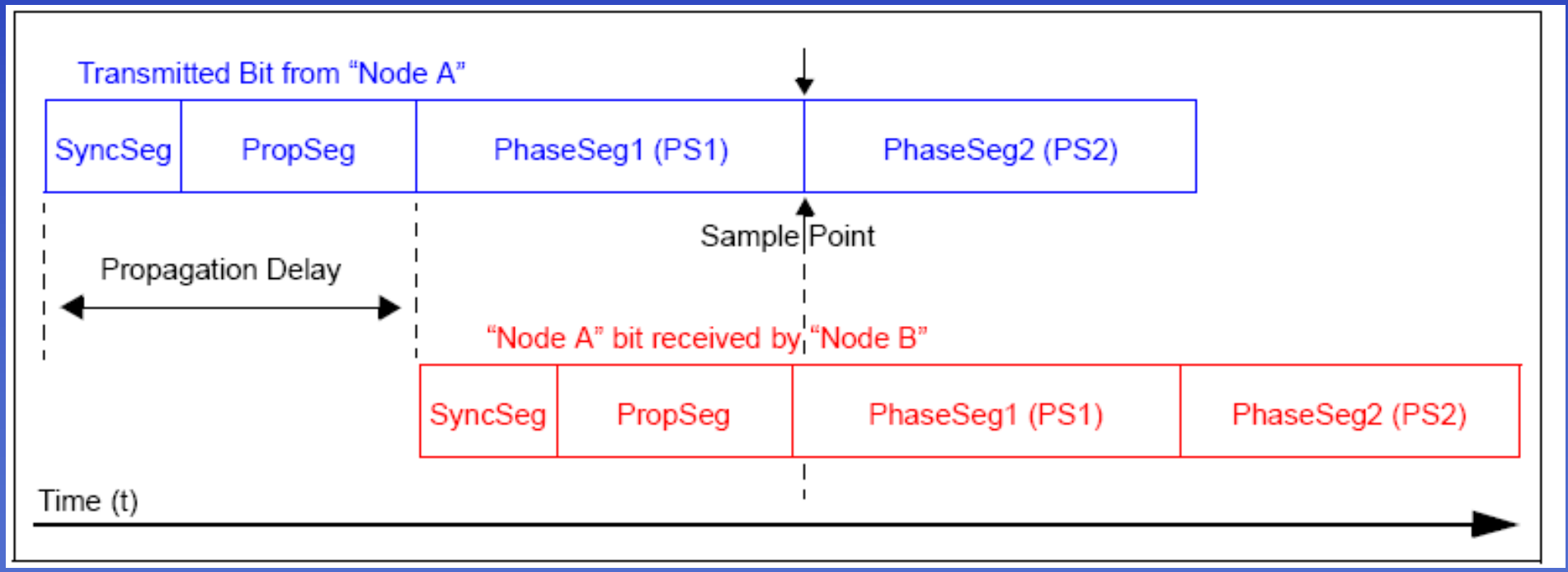
Length (meters)	(Bit per Second)
40m	1Mbps
500m	125Kbps
1,000m	50Kbps



- **CAN Network 的規模**
  - 2.0B 版本的通信協定有能力支援最多達 2^{29} 個 node

CAN BUS
LENGTH -v- BIT RATES

Key Element : Propagation Delay



CAN BUS LENGTH -v- BIT RATES

- CAN belongs to a group of protocols known as - 'In Bit Response' protocols
  - Requires that bit time to travel from transmitting node to node farthest away shall take no longer than 2/3 of total bit time
    - Result is a reduction in bus length with an increase in bit rate
  - Remaining 1/3 of total bit time allows
    - Receiving node to perform bit wise arbitration
    - Switch into receive mode if arbitration lost

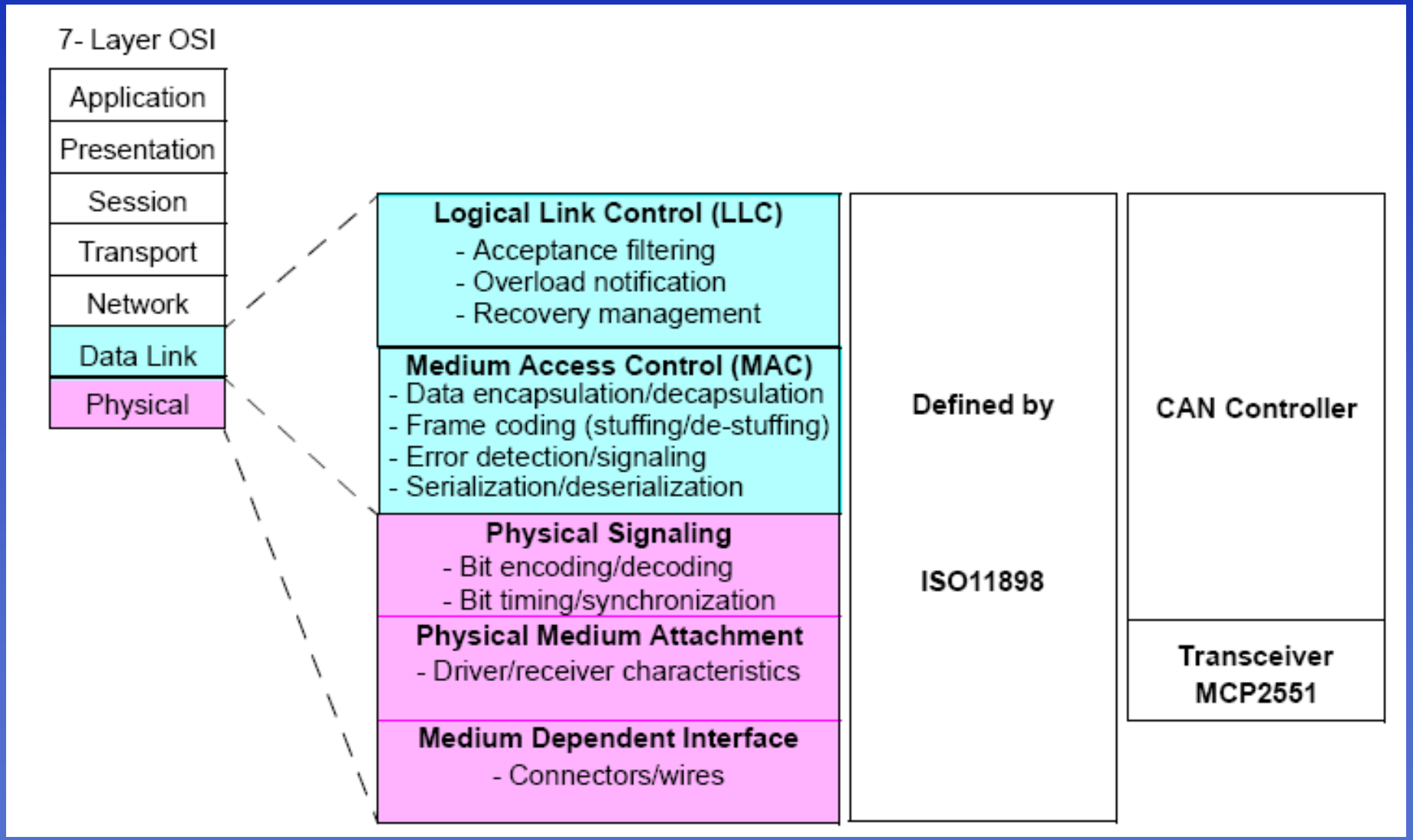
CAN BUS LENGTH -v- BIT RATES

- Calculating Max. Transmission Distance for Bit Rate
  - Factors
    - Speed of electrical wave in copper is approx 2/3 of speed of light in a vacuum = 2/3 * 30cm/ns = 17cm/ns ($t_{prop\text{-}line}$)
    - Average propagation delay in transceiver = 25ns ($t_{prop\text{-}tcvr}$)
  - Equation - First step approximation
    - $2/3\ t_{bit} >= 4 * t_{prop\text{-}tcvr} + 2 * t_{prop\text{-}line}$
  - Results in 'rule of thumb' design rules
    - 40m at 1Mbit/s (CAN max. bit rate)
    - 400m at 100Kbit/s
    - 1000m at 40Kbit/s
    - Values may change depending on choice of transceiver

CAN BUS
CAN node Number definition

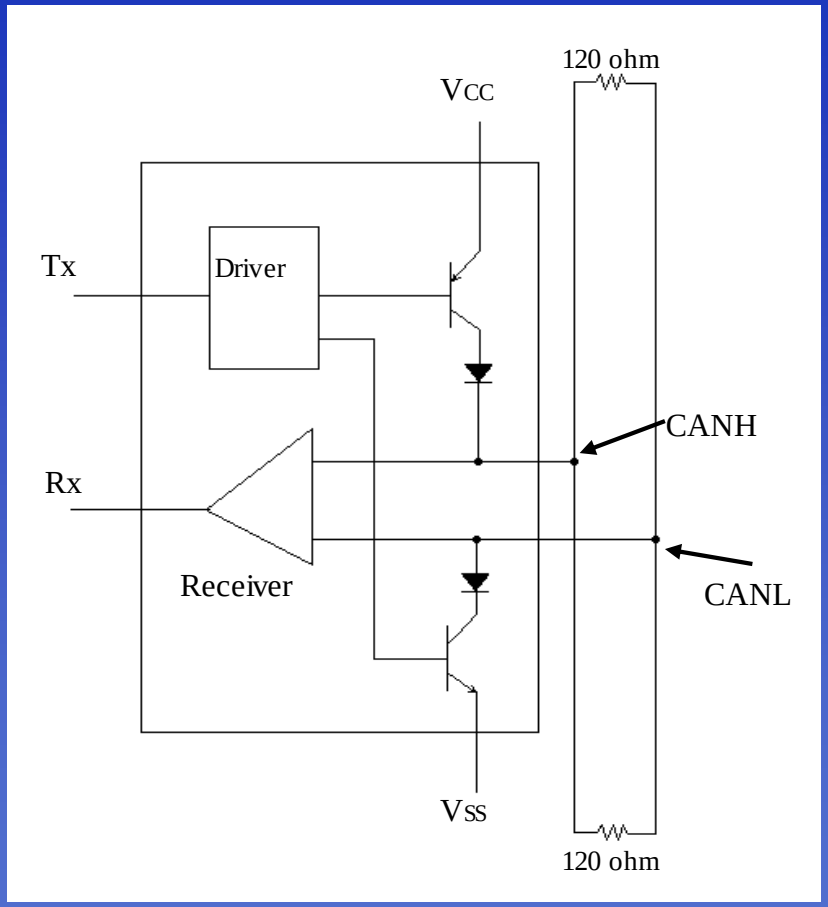
- Number of Nodes in a System
  - Factors
    - Electrical Characteristics of transceiver
    - Application specific drivers
  - Real Systems
    - Standard Differential Line Drivers - 32 Nodes
    - High Speed Diff. Line Drivers eg. MCP2561/2 =112 Nodes

What ISO 11898 Defined vs. solution



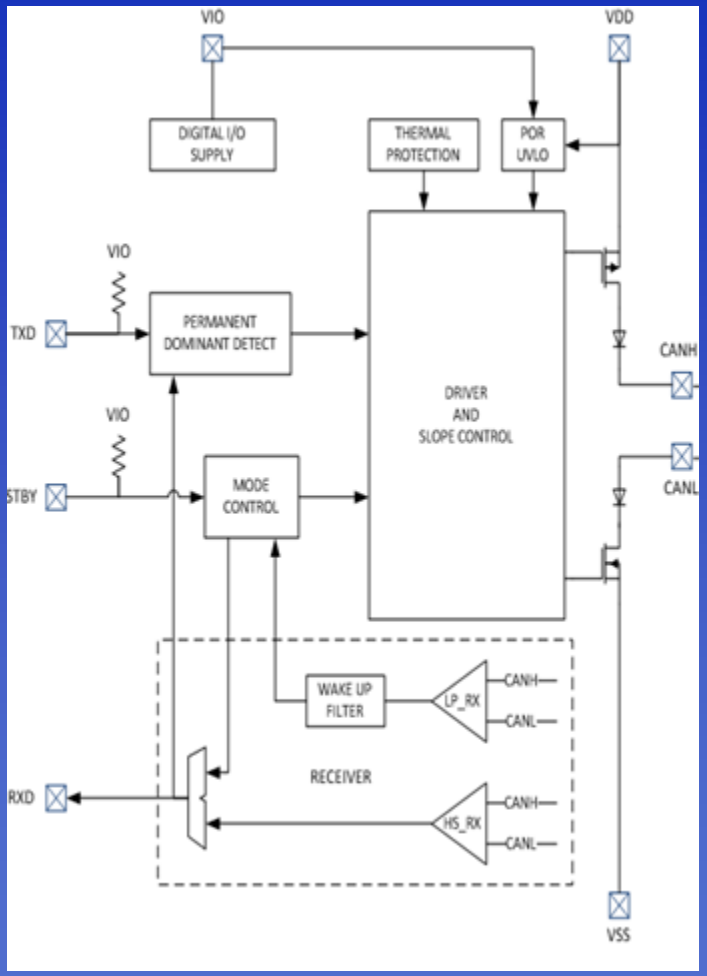
CAN 5V Differential Physical Layer

- 通常被稱為 CAN transceiver (例如: MCP2551/MCP2561/MCP2562)
- Vcc 由 4.5 to 5.5V
- 與 uC 界面的 Tx 和 Rx 皆為數位的輸出/輸入 信號
- CANH 和 CANL 間差動信號的電壓為 0V 到 3.0V
  - ✓ ΔV > 0.9V 稱為 dominant
  - ✓ ΔV < 0.5V 稱為 recessive
- Up to +/-40V continuous capable on CANH and CANL pins
- +/- 200V transient capable on CANH and CANL
- 40m max cable length @ 1Mbps

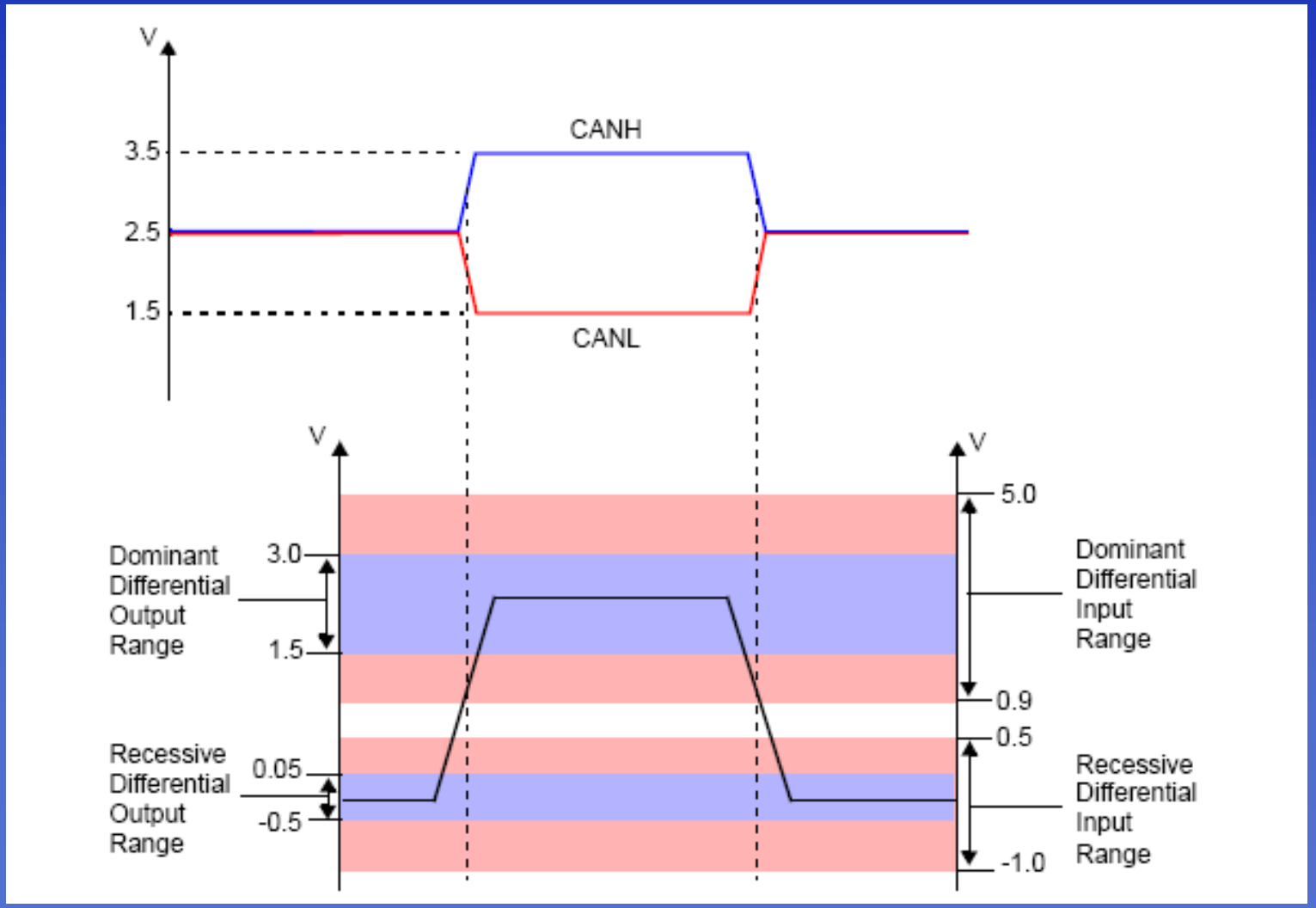


CAN High Speed Physical Layer

- CAN FD transceiver (ATA6562/63)
- ISO11898-2:2016 compliant
- Supports up to 5Mbps
- VDD 4.5V to 5.5V
- VIO - Supply voltage for I/O level adaptor
- TXD and RXD are digital I/O to the uC
- CANH and CANL differential bus
  - DV > 0.9 V is dominant (0)
  - DV < 0.5V is recessive (1)
- Max rating -27 to 42V DC voltage on CANH and CANL
- Max rating -150V to 100V transient Voltage on CANH and CANL



ISO11898 Nominal Bit Level



Comparing MCP2551 to ISO11898

Parameter	ISO-11898-4		MCP2551		Unit	Comments
	min	max	min	max		
DC Voltage on CANH and CANL	-3	+32	-40	+40	V	Exceeds ISO-11898
Transient voltage on CANH and CANL	-150	+100	-250	+250	V	Exceeds ISO-11898
Common Mode Bus Voltage	-2.0	+7.0	-12	+12	V	Exceeds ISO-11898
Recessive Output Bus Voltage	+2.0	+3.0	+2.0	+3.0	V	Meets ISO-11898
Recessive Differential Output Voltage	-500	+50	-500	+50	mV	Meets ISO-11898
Differential Internal Resistance	10	100	20	100	kΩ	Meets ISO-11898
Common Mode Input Resistance	5.0	50	5.0	50	kΩ	Meets ISO-11898
Differential Dominant Output Voltage	+1.5	+3.0	+1.5	+3.0	V	Meets ISO-11898
Dominant Output Voltage (CANH)	+2.75	+4.50	+2.75	+4.50	V	Meets ISO-11898
Dominant Output Voltage (CANL)	+0.50	+2.25	+0.50	+2.25	V	Meets ISO-11898
Permanent Dominant Detection (Driver)	Not Required		1.25	—	ms	
Power-On Reset and Brown-Out Detection	Not Required		Yes		--	

CAN Protocol Basics

CAN 的主要優點與特性

- 使用如下的通信協定來增加傳輸的效能
- Carrier Sense Multiple Access and Collision Detection with Collision Resolution (CSMA/CD-CR)
- 以 Message 為信息傳遞的依據而非位址 (Address)
  - CAN Node 可以要求對方立刻傳送資料出來(發出 Remote Transmit Request 的要求), 與一般傳送的資料封包格式完全一樣, 其差別只在於 RTR 位元的值為 “1”
- ** CAN 是一種快速, 可靠的通信方式

CSMA/CD-CR

CS Carrier Sense - 在每次開始資料的傳送之前, 每一個 CAN node 必需偵測 bus 上的動作並確定一段期間內無任何的信號活動

MA Multiple Access - 當持續性的一段時間內無信號活動, 每一個 node 都有相同的機會來傳送訊息到 CAN bus

CD Collision Detection - 如果兩個 node 在同一時間點送出訊息, 則衝突便會發生. CAN 必需能偵測出此衝突的事件

CR Collision Resolution - 使用非破壞式的逐位元仲裁

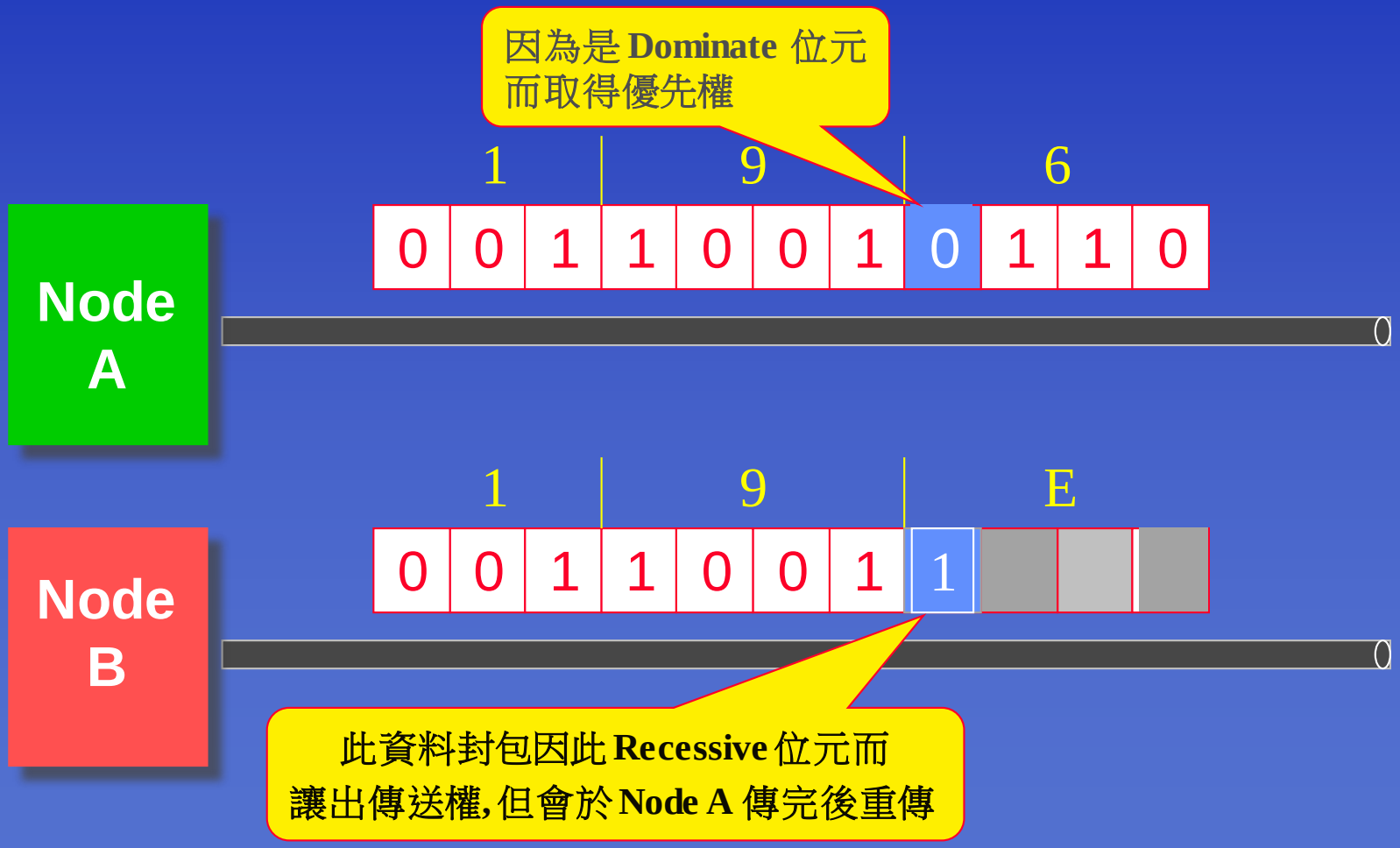
CSMA/CD-CR

CR Collision Resolution - 非破壞性的逐位元仲裁

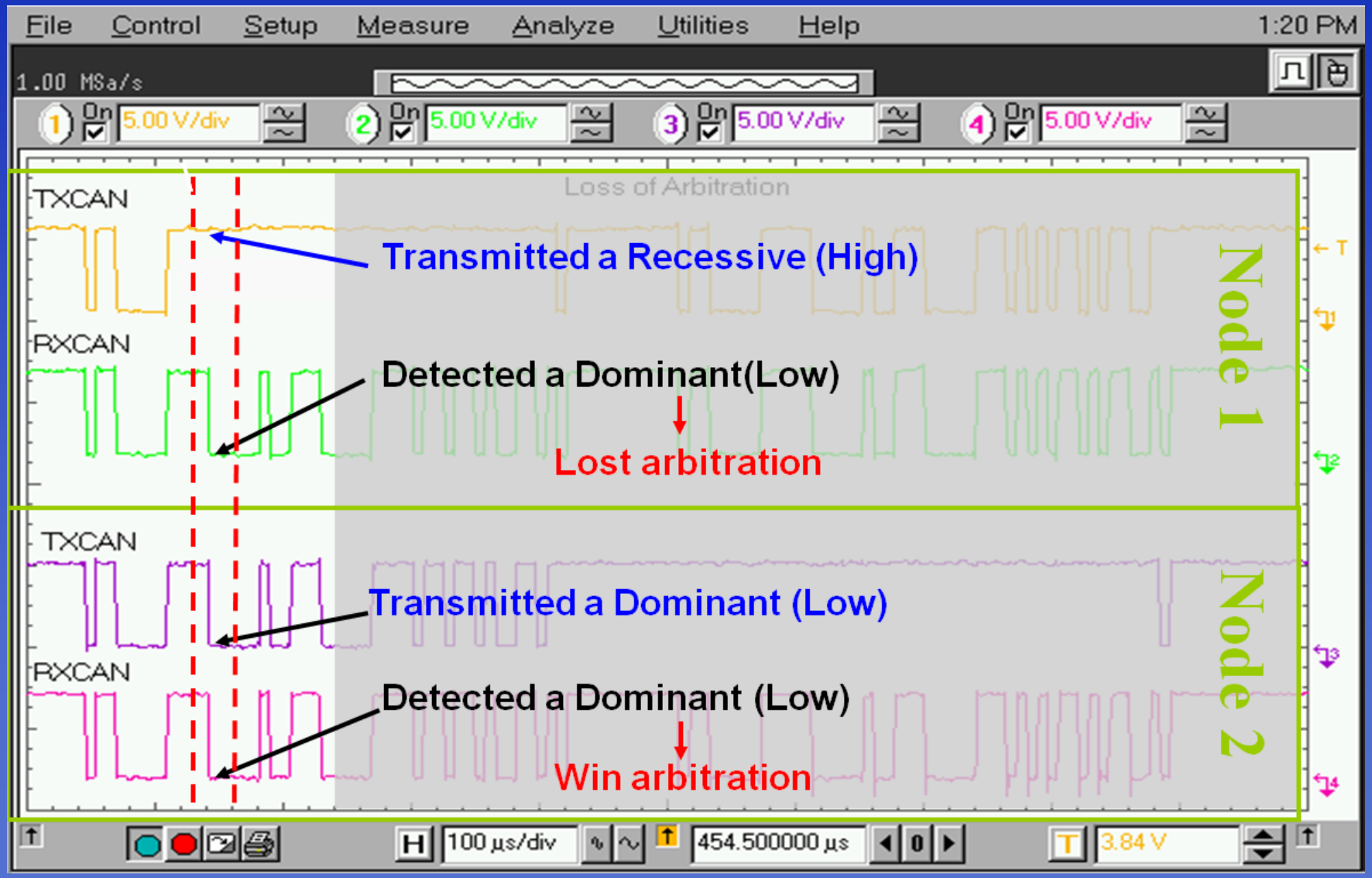
- 即使資料碰撞發生, 所要發送的 Messages 仍然保持其完整性
- 所有仲裁行為發生時都不會對具有最高優先權的資料造成任何延遲或者錯誤
- 任何因為優先權較低而無法在第一時間傳送的資料封包會自動於下一個可傳送的時間自動的被傳送
- 要達到上述功能的需求:
  - ✓ Dominant 以及 recessive 位元的狀態必需被事先定義
  - ✓ 每個 node 必須不斷的監視 bus 上的資料流, 比較自己送出的資料與真正出現在 bus 上的資料是否相符 !

CSMA/CD-CR

CR Collision Resolution - 非破壞性仲裁的例子



Arbitration(仲裁) 的信號範例



CAN is Message-Based (Not Address-Based)

- Messages 並非以位址為依據, 由一個節點被傳送到另一個節點
- 在傳送的訊息中包含了資料本身以及優先等級等相關資訊
- 所有的節點 (nodes) 對每個傳輸的資料都會接收, 並在收到正確的封包後回應 ACK (內建於 CAN 周邊的硬體功能中)
- 各個 Node 自行決定要處理此資訊或置之不理 (可經由軟體或設定適當的 Masks 以及 Filters 來達成)
- 因為使用的是 message-based 的架構, node-to-node 以及 broadcast (欲由一個以上的 node 接收並處理) 的訊息皆可經由相同的訊息(message) 來傳送
- 新的 nodes 可隨時被加入系統中, 其他 nodes 並不需因為此新 node 的加入而必需更新內部資訊

CAN 是以 Message 資訊基礎的通信 (非傳統以 Address 的點對點方式)

Message Frames

- Data Frame
  - 最為普遍的類型
  - 資料被從一個節點傳送到所有的節點

- Remote Frame
  - 與 Data Frame 極為相似 (只有 RTR bit 被設為 1)
  - 被用來向其它的 CAN node 主動要求資料之用

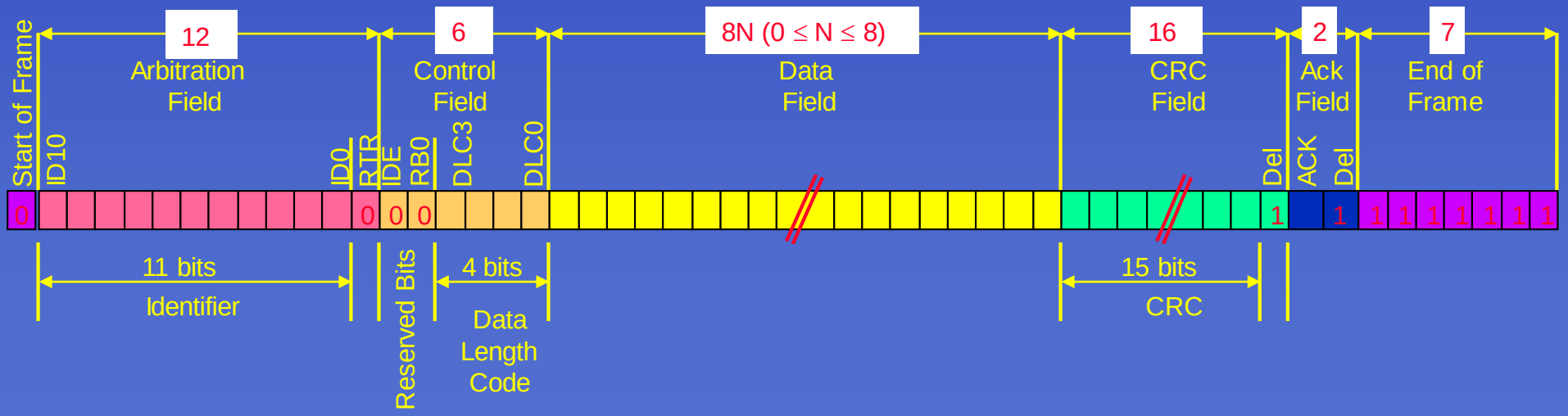
- Error Frame
  - 用來表示在 CAN bus 上有錯誤狀況被偵測到

- Overload Frame
  - 用來告知其他的 CAN nodes, 必須有更多的時間來處理收到的資料. 主要的功能是延遲下一個封包被傳送的時間

Data Frame

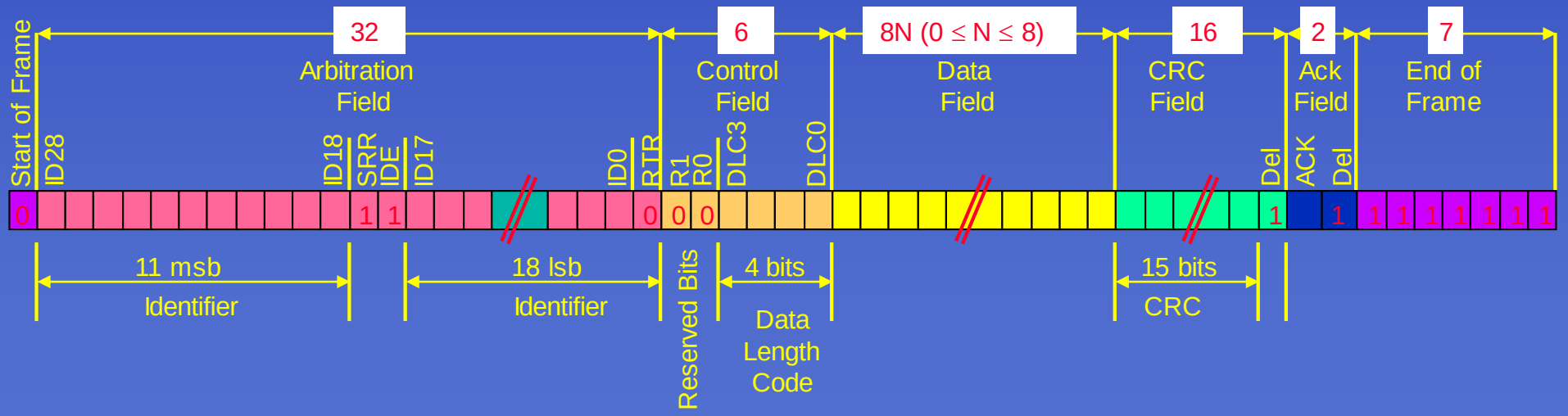
有兩種型式的 Data Frame, 依據不同版本的 CAN 協定規範而有差異, 但新版本可相容於舊的版本

- **Standard Data Frame**
  - 依據較早的 CAN 規範所訂定 (versions 1.0 及 2.0A). 定義成具有11 個位元的辨識欄位(identifier field)



Data Frame

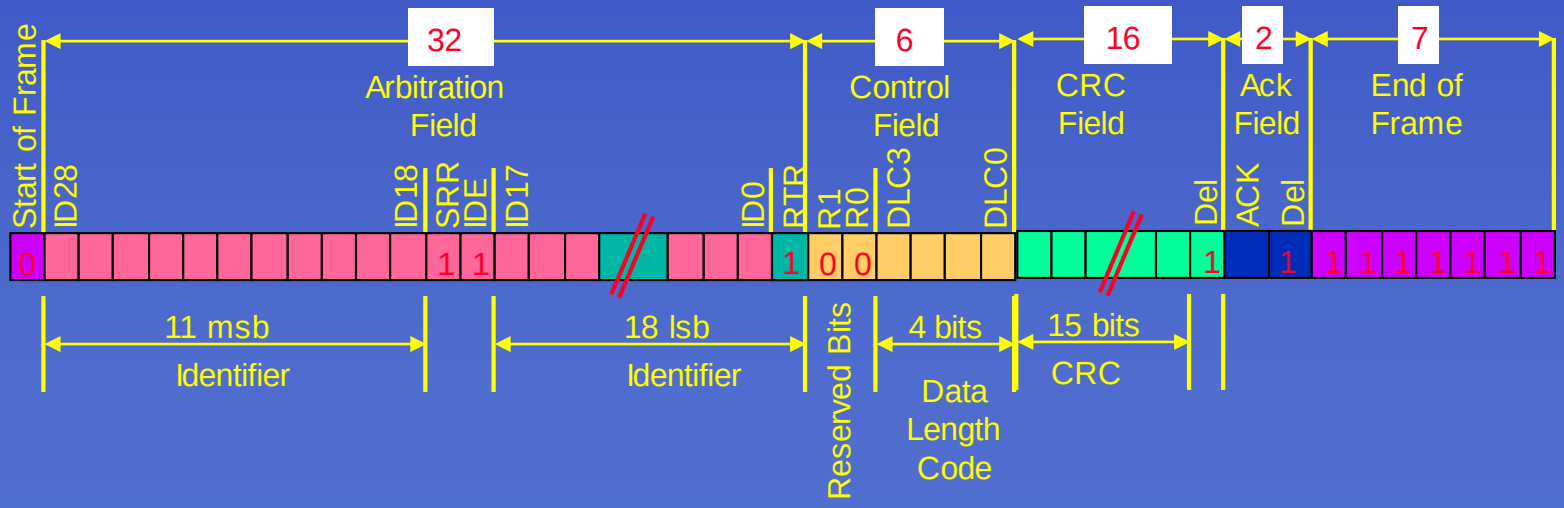
- Extended Data Frame
  - 依據最新的 CAN 協定規範所定義 (version 2.0B). 辨識欄位(identifier field) 被擴展為 29 位元; 增加了可用 Message 的數量



Remote Frame

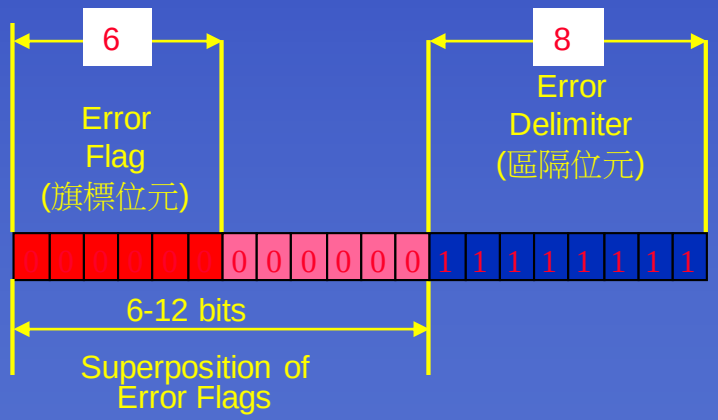
- 與 Data Frame 一樣, 有兩種型式的 Remote Frame, 依據不同版本的 CAN 協定規範而有差異, 但新版本可相容於舊的版本

- 除了不具備資料與 RTR 位元被設定為 "recessive" 狀態外, 與 Data Frame 幾乎完全相同



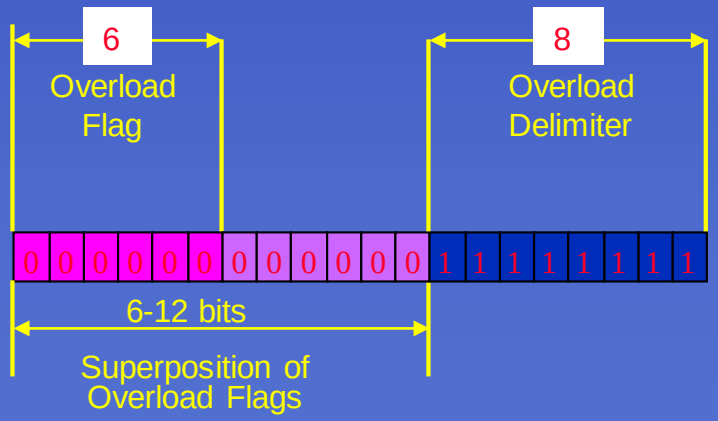
Error Frame

- Error Frame
  - 當 CAN node 偵測到 CAN 所定義的眾多型態的錯誤時, 將產生 Error Frame
  - Error Frames 將在後續的說明中描述



Overload Frame

- Overload Frame
  - 當一個 node 需要較多的時間來處理所接收的資料時, 便送出 Overload Frame
  - 當間隔位元少於規定值時 (最少要 3 個 recessive bits) 也會發生
  - 間隔位元 (Intermission) 使 CAN node 有時間做內部資料處理



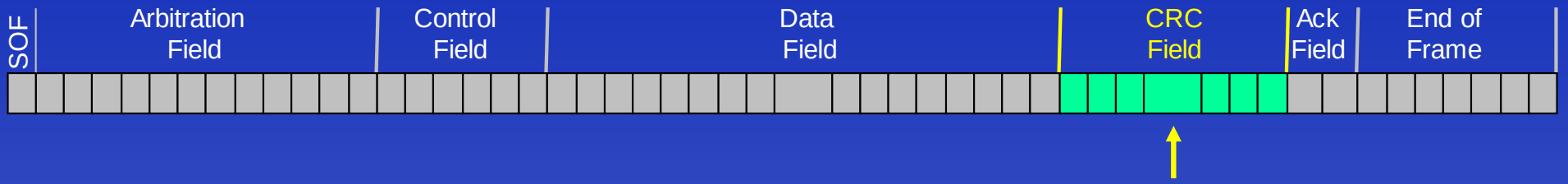
CAN 對 Error 的處理

- 為了正確的傳遞資訊, CAN 的通信協定非常嚴格地定義了許多錯誤的狀況, 當在不同的偵測點偵測到錯誤時皆會做出反應(根據 CAN 控制器當時的 Error State)
  - 確保 messages 的完整性

- 必需對有缺陷的 node 做處置並制約其行為(Fault Confinement)
  - 依據錯誤程度的不同, CAN 的各 node 可能由正常的工作模式轉而進入完全離線的狀態
  - 對有錯誤或缺陷的 node 做出限制可防止有缺陷的 node 因為不斷做出無效的傳送而使整個網路動彈不得

CAN 可偵測的各種錯誤

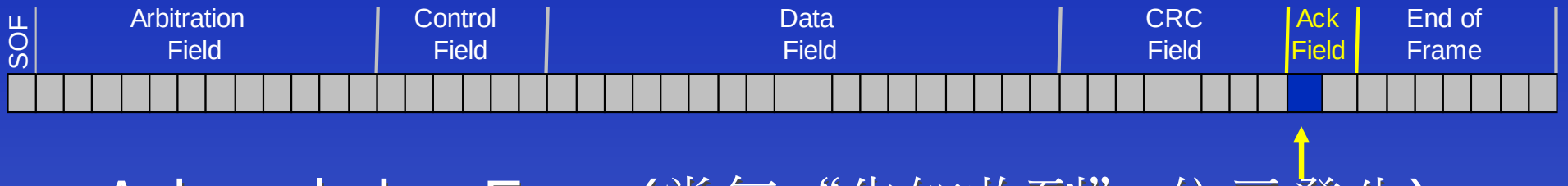
CAN Message



- CRC Error
  - 15-bit 的 CRC 會自動的被加入被傳送中的 Message 之 CRC 欄位中
  - 所有 node 皆接收 Message, 並計算 CRC 後與接收到的 CRC 資料相比對
  - 若兩者 CRC 不相等則視為發生 CRC 錯誤 並且產生一個 Error Frame
  - 傳送此 message 的 CAN node 偵測到此錯誤後將重新傳送原來的 Message

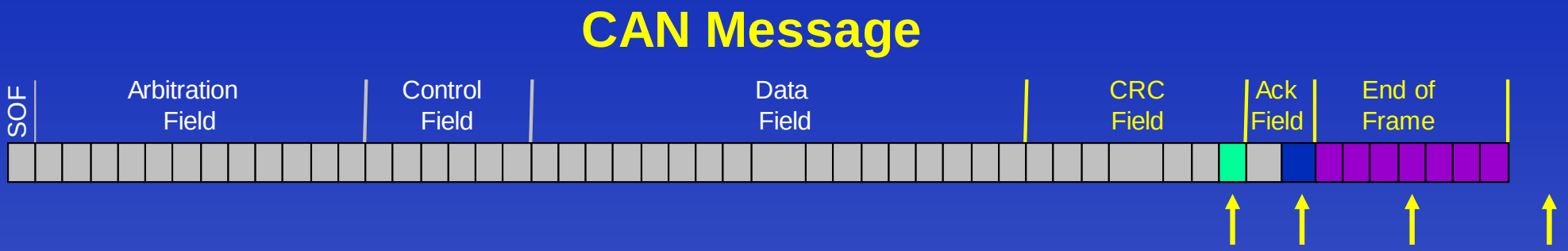
CAN 可偵測的各種錯誤

CAN Message



- Acknowledge Error (當無 “告知收到” 位元發生)
  - 傳送中的 node 在 Ack Slot 時檢查 Ack 位元, 此時它送出一個 recessive 位元並檢查是否有收到 dominant 位元
  - 如果偵測到 dominant 位元發生, 表示至少有一個 node 已正確地收到 Message
  - 否則, 將視為有 Ack 錯誤發生, 將產生一個 “示誤封包(Error Message)” 並重新傳送此次的資料

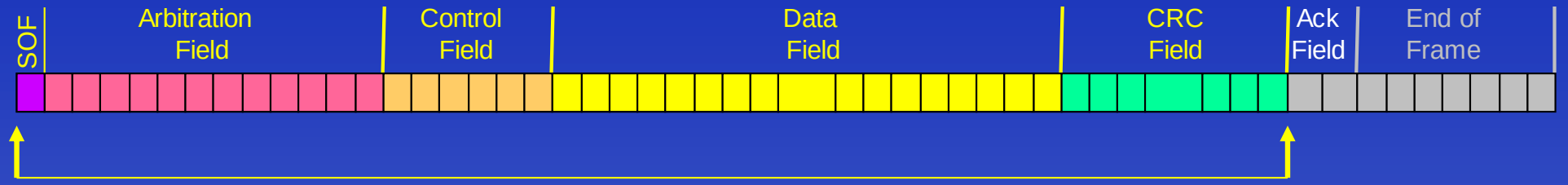
CAN 可偵測的各種錯誤



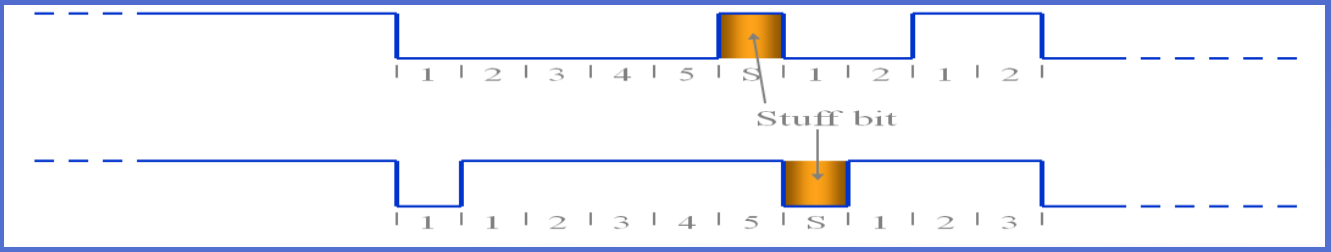
- Form Error
  - 若任一個 node 在 CRC Delimiter, Ack Delimiter, End of Frame (EOF) field 或 Interframe 的間隔偵測到有 dominate 位元則產生一個 Error Frame 來指明發生了 Form Error
  - 原先的 message 將在 Error Frame 結束後重送

CAN 可偵測的各種錯誤

CAN Message

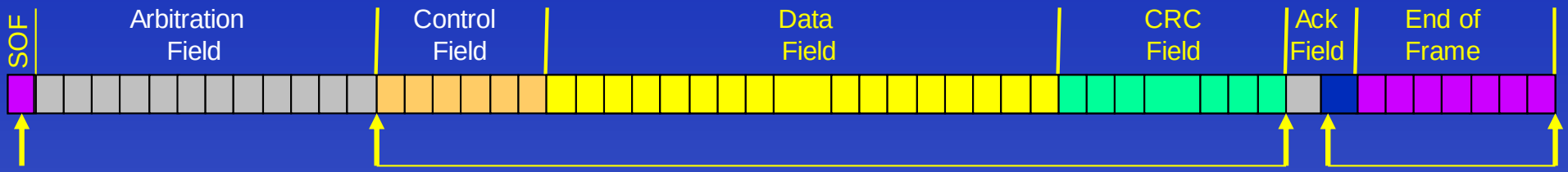


- Stuff Error
  - CAN 的協定明確的定義，不可以有超過 5 個狀態相同的 bit 連續發生, 若有要於 5 個相同的 bit 後補一個反相的 bit
  - 如果有 6 相同的 bit 在 SOF 以及 CRC Delimiter 間連續發生，則被視為違反了位元填充 (bit Stuffing) 的原則, 將產生 Error Frame 來回應偵測到的錯誤
  - 原先的 message 將在 Error Frame 結束後重送



CAN 可偵測的各種錯誤

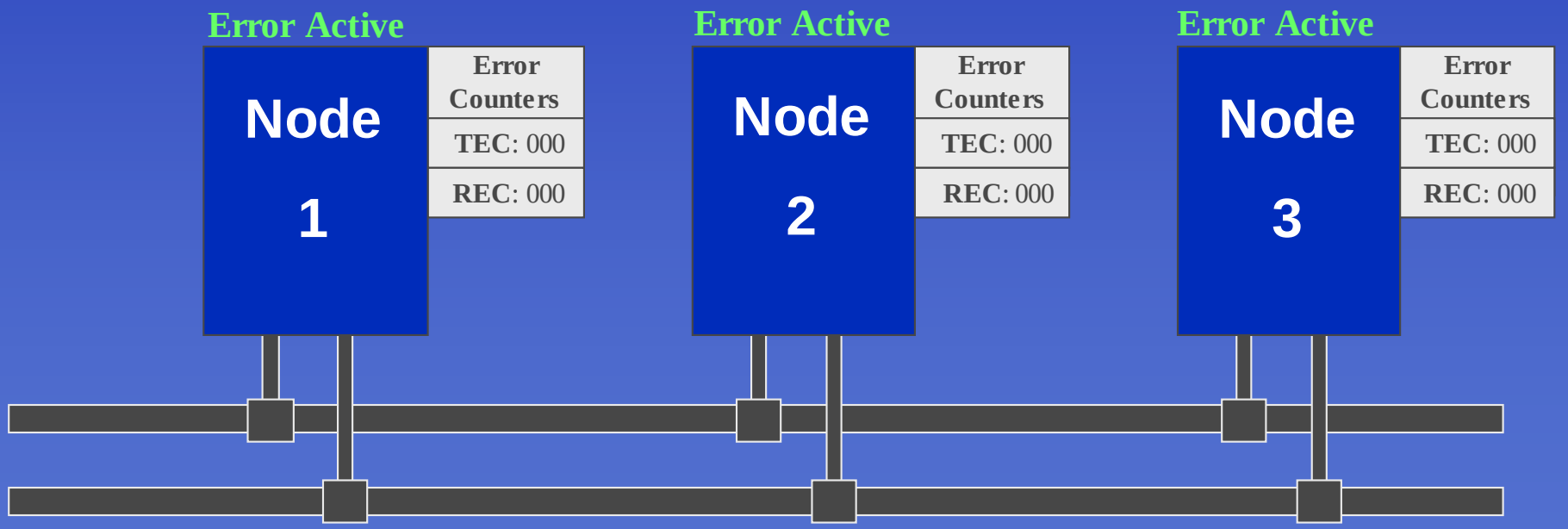
CAN Message



- Bit Error
  - 當傳送端發現它送出的信號與實際出現在 CAN Bus 上的不同, 則判斷有 Bit Error 發生
  - 例外狀況 :
    - 仲裁 (arbitration) 的區間不會發出 Error Frame （ 標準的仲裁程序)
    - 在 Ack bit 的區間 (因為傳送端會送出 recessive 而 ACK 端要送出 dominate - 標準的 acknowledge 程序)

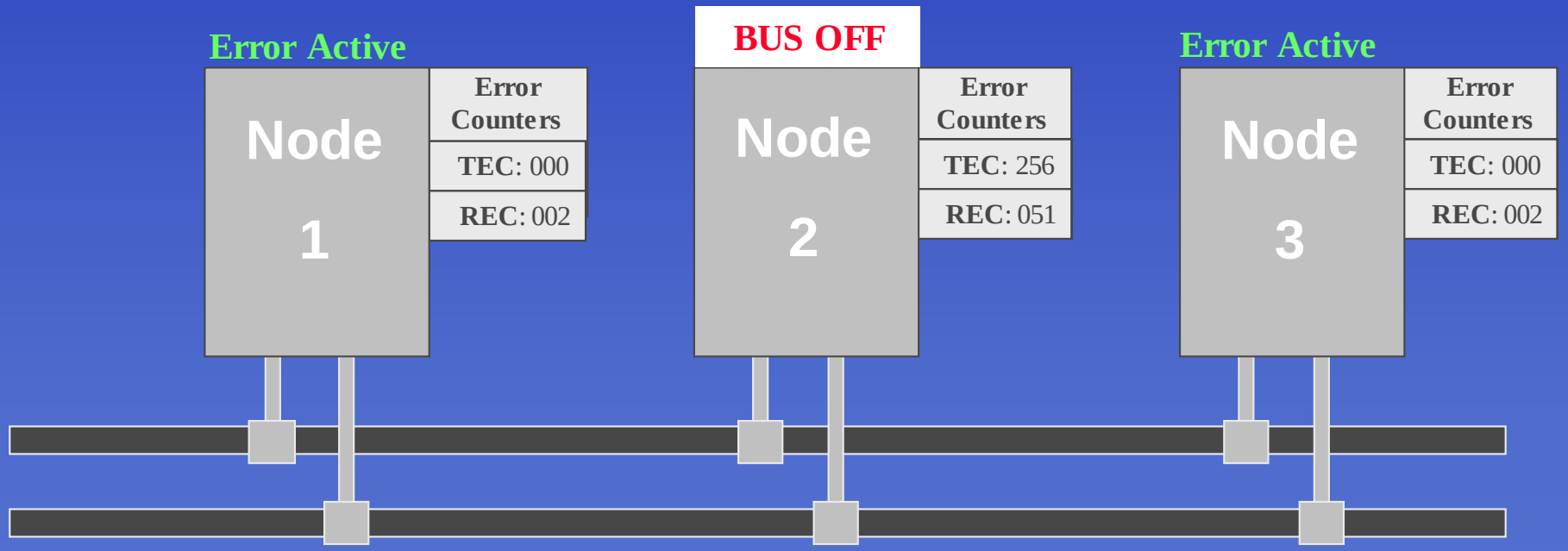
CAN 對錯誤的採取的限制

- 每一個 CAN node 都有各自獨立的 傳送以及接收錯誤計數器 (transmit 以及 receive error counters) TEC 與 REC
- CAN 協定根據 Error Counter 的內容將錯誤狀態分為三種： Error Active 、Error Passive、 Bus-Off



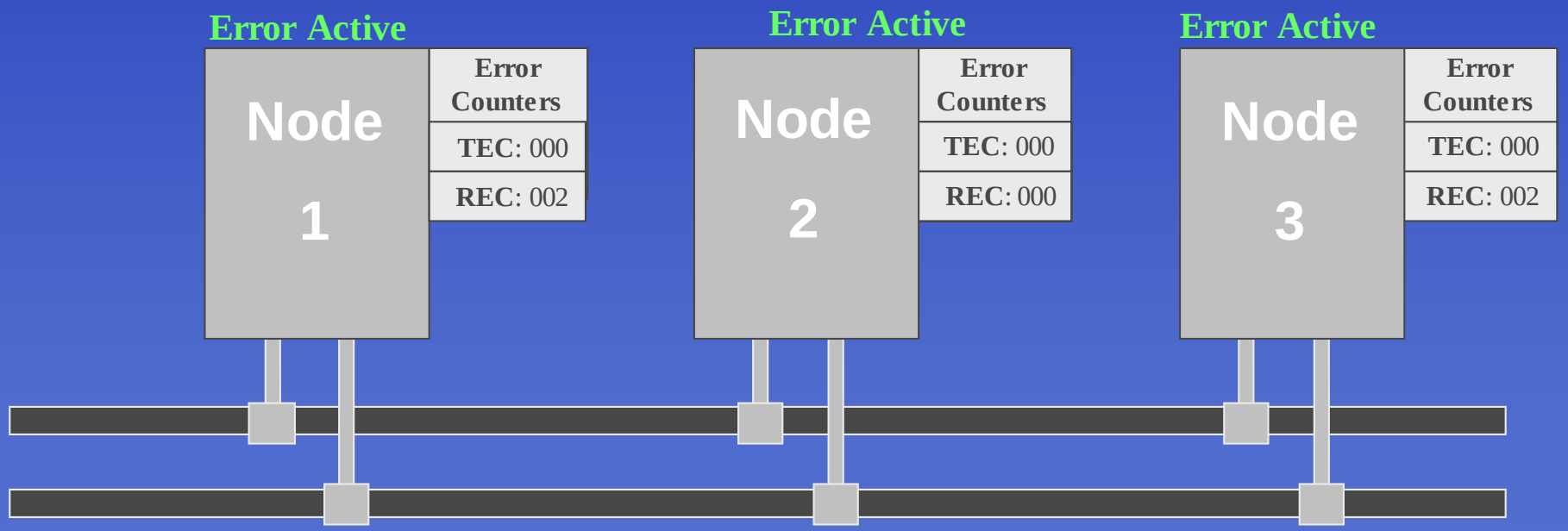
CAN 對錯誤的採取的限制

- 根據 node 所處的狀態不同, 其控制 CAN bus 的能力也跟著調整 (Active > Passive > Bus-Off)
- 如果到達了 Bus-Off 的狀態條件 (TEC > 255, CAN node 可將自身設定為完全離線狀態



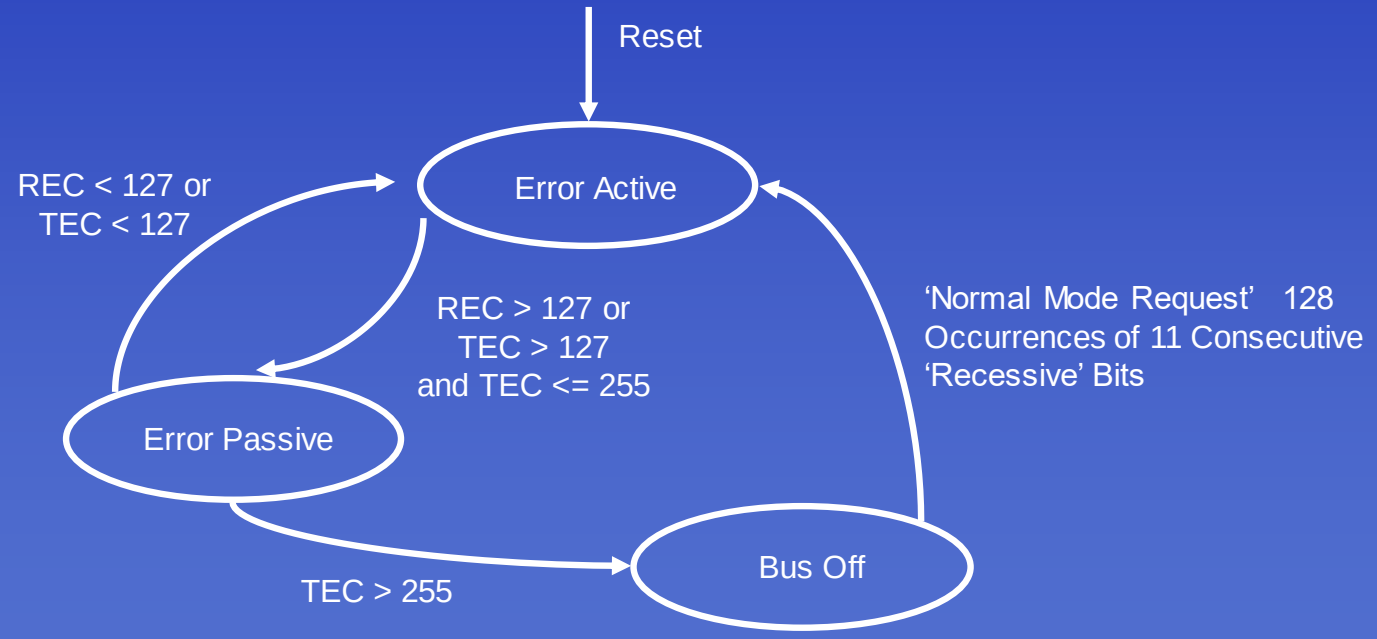
CAN 對錯誤的採取的限制

- 對已經進入 Bus-Off 狀態的 node；CAN 定義了復原 (recovery) 的程序以便使其重回到活動(Active) 狀態
- 如此的做法可以防止嚴重錯誤的 node 持續的佔用有限的網路頻寬



CAN BUS ERROR HANDLING

- Error States
  - Error Modes - State Transition Diagram



Error-Active (較積極的錯誤回應)

- CAN node 可以主動地傳送或接收訊息(messages) 和 Error-Frames (一般正常的 工作模式)
- 當偵測到錯誤時, CAN node 將以送出正常 Error Frame (使用 6 個 dominate 位元的 Active Error Flag)的方式將正傳送中的 message 中斷掉
- 這樣的做法違反了位元填充的原則, 所以其他的 CAN node 也將產生它們自己的 Error Frames (稱為錯誤回應旗標 -> Error Echo Flags)
- 每當偵測到有錯誤發生, 適當的錯誤計數器 (TEC 或 REC) 將被依當時的狀態累加
- 一旦Error Frame 完成後, bus 的活動又回到正常狀態
  - Message 將會被重新傳送 !!

Error-Passive (消極的錯誤回應)

- 當 TEC 或 REC 計數器超過了127, CAN node 變進入 **Error-Passive 的狀態 (採用較消極的錯誤回應)**
- 當一個處於 Error-Passive 狀態的 node 偵測到錯誤發生時將送出一個用 Passive Error Flag (6 個 recessive 位元)組成的 Error Frame
- 使用 Passive Error Flags 的 Error Frames 將不會影響傳送中的 Message
- 適當的錯誤計數器 (TEC 或 REC) 將被加累加
- 當 Error Frame 完成後, 處於 Error-Passive 的 node 會等到以下的條件符合後才會被致能傳送
  - 在 dominate 位元之後有 3 個 intermission 的位元加上 8 個位元的時間長度

Bus-Off 狀態

- 當 TEC 計數器值超過 255
  - 控制器將進入 Bus-Off 狀態
- 此時的 CAN node 將無法傳送或接收任何的 message 或錯誤狀態
- 經過以下的程序, CAN node 可再度回到 Error Active 的狀態
  - 接收到正確的 “bus recovery” 程序
    - 11 個連續的 recessive 位元發生128 次 !!
  - 經由硬體或軟體的重置動作
  - 將控制器設定成 “Configuration mode”

錯誤偵測的的重點整理

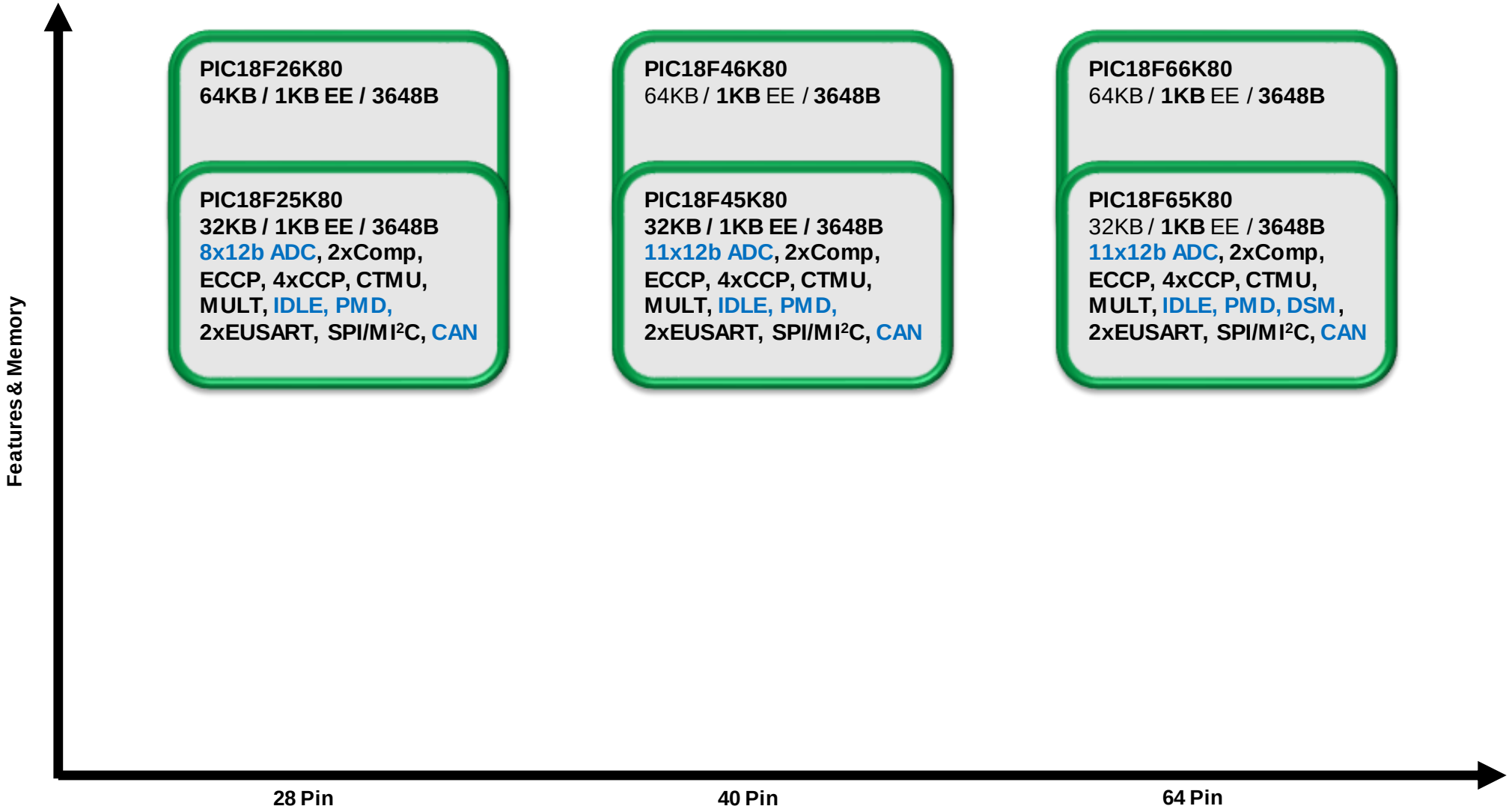
- CAN nodes 可依錯誤的嚴重程度來自動地減少它們對 bus 的控制能力
- 透過以下三種狀態的轉換可以達成這樣的要求
  - Error-Active (正常的操作模式)
  - Error-Passive (控制 bus 的能力被降低)
  - Bus-Off (無法控制 CAN bus)
- 不同 Error-States 間的差異在於錯誤的 node 該如何被限制行為
- CAN 對錯誤時的能力限制 (Fault Confinement) 可防止有缺陷的 node 使得整個網路當掉

Microchip CAN Solutions

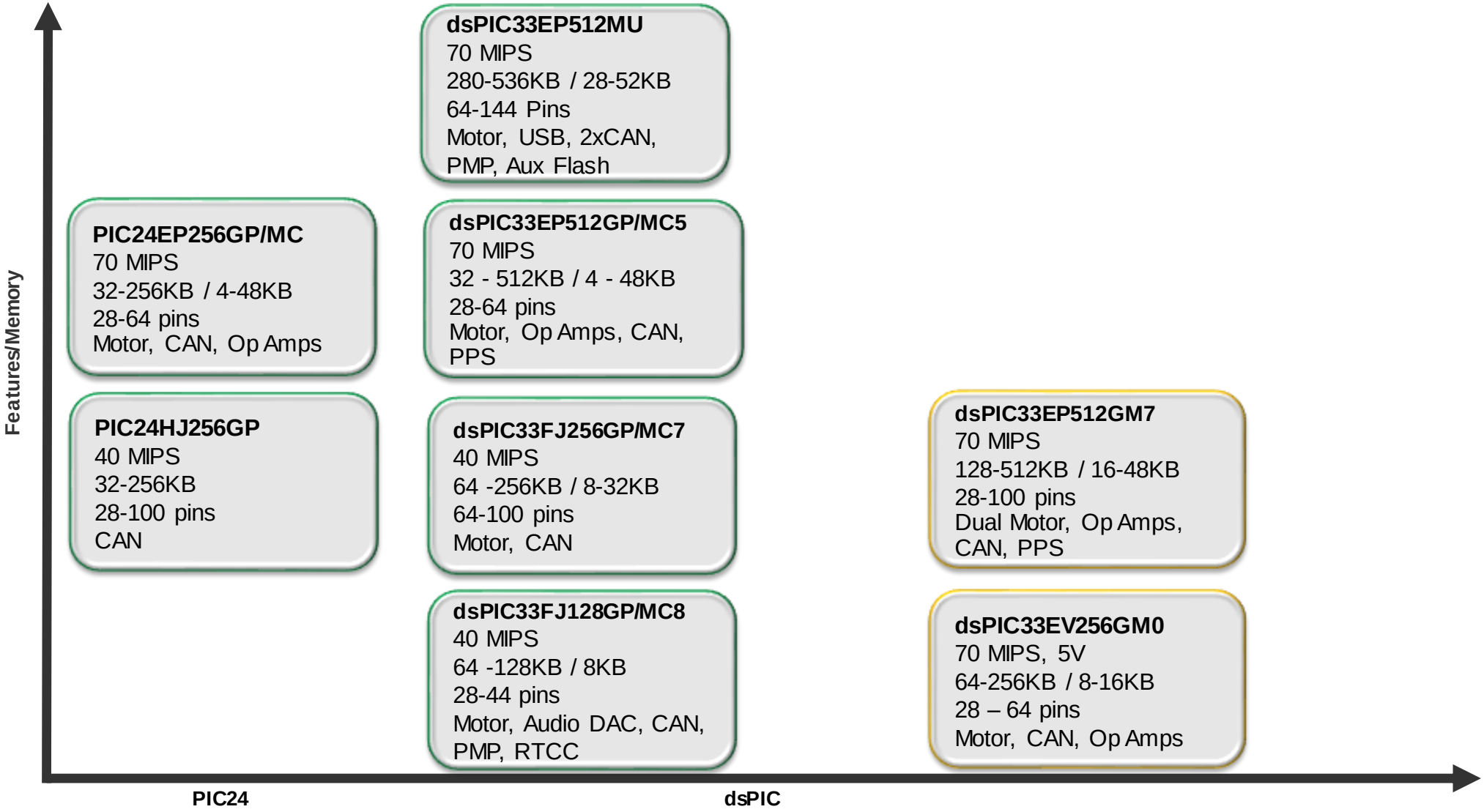
PIC18Fxx80 系列
內含 CAN/ECAN Controller 的產品

PIC18(L)FxxK80

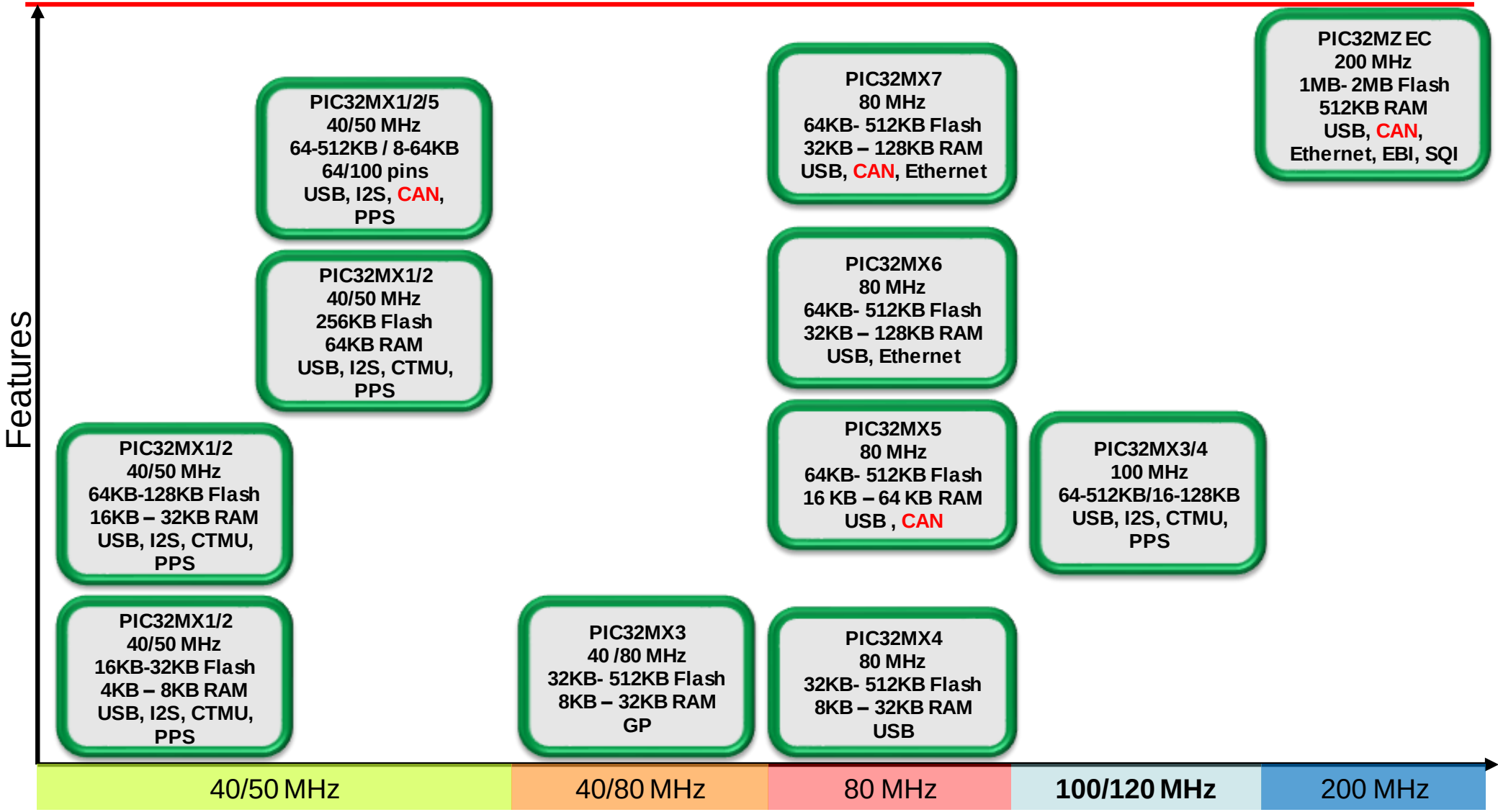
Integrated CAN Controler & 12b ADC



16-bit MCU with CAN



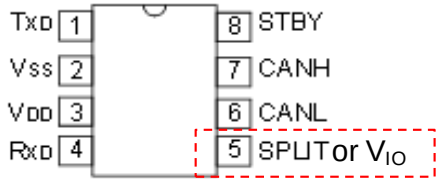
PIC32 Family



CAN Transceiver and Peripheral Products

MCP2561/2

High-Speed CAN Transceivers



Automotive Grade[1]

CAN / ISO 11898 Compliant

SPLIT Option (common mode stabilization)

VIO Option (internal level shifting)

CANH/L ESD ≥ 14kV

Standby current < 5uA (Typ)

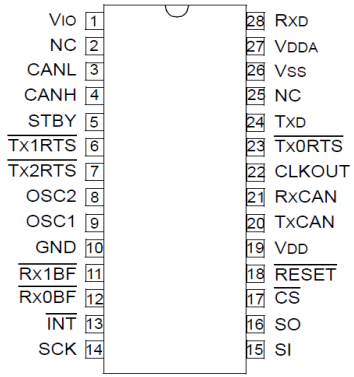
-40˚C to +150˚C

DFN/SOIC/PDIP 8L packages

[1]Meets OEM Hardware Requirements for LIN, CAN and FlexRay Interfaces in Automotive Applications", Version 1.3, 2012

MCP25625

CAN Controller w/HS CAN Transceiver



Automotive Grade

CAN 2.0 up to 1 Mb/s Operation

Very Low Standby Current (10 µA, typ)

Up to 10 MHz SPI Clock Speed

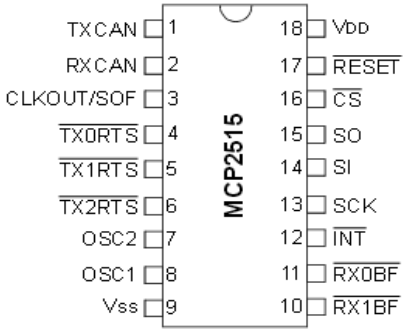
Interfaces to MCUs with 2.7V to 5.5V I/O

Available in SSOP-28L & 6x6 QFN-28L

-40˚C to +125˚C

MCP2515

Stand-alone CAN Controller



CAN V2.0B at 1 Mb

2 receiver buffers, 6 filters and 2 masks

SPI interface up to 10MHz

Vdd 2.7V to 5.5V

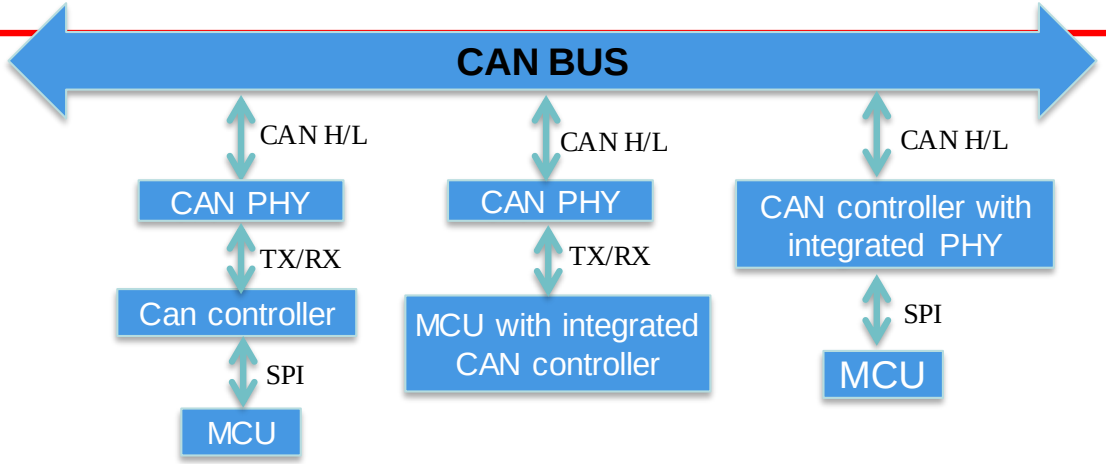
Sleep mode current < 1uA

-40˚C to +125˚C

18L SOIC/PDIP, 20L TSSOP/QFN

ISO-16845 DIS CAN conformance

Microchip CAN & CAN FD Network Solutions



Microchip Solution	Suggested Products	Descriptions
CAN transceiver (PHY)	• ATA6560/61 • ATA6562/63/64 • ATA6565(Dual CAN Trx) • ATA6570 (Partial Networking)	• CAN classic and CAN FD compliance
CAN controller	• MCP2515 • MCP2517FD	• SPI to CAN classic bridge
CAN controller with integrated transceiver	• MCP25625	

MCU with integrated CAN controller	• 8-bit/16-bit/32-bit MCUs with ECAN module(s)
	  • PIC18FxxK8x (PIC18F66K80)
	  • dsPIC33F, dsPIC33E
	  • PIC24H, PIC24E
	  • PIC32MX, PIC32MZ
	• SAM V7x with CAN FD controller
	• SAMC21	

Exercise 0
CAN Bus Monitor Introduction

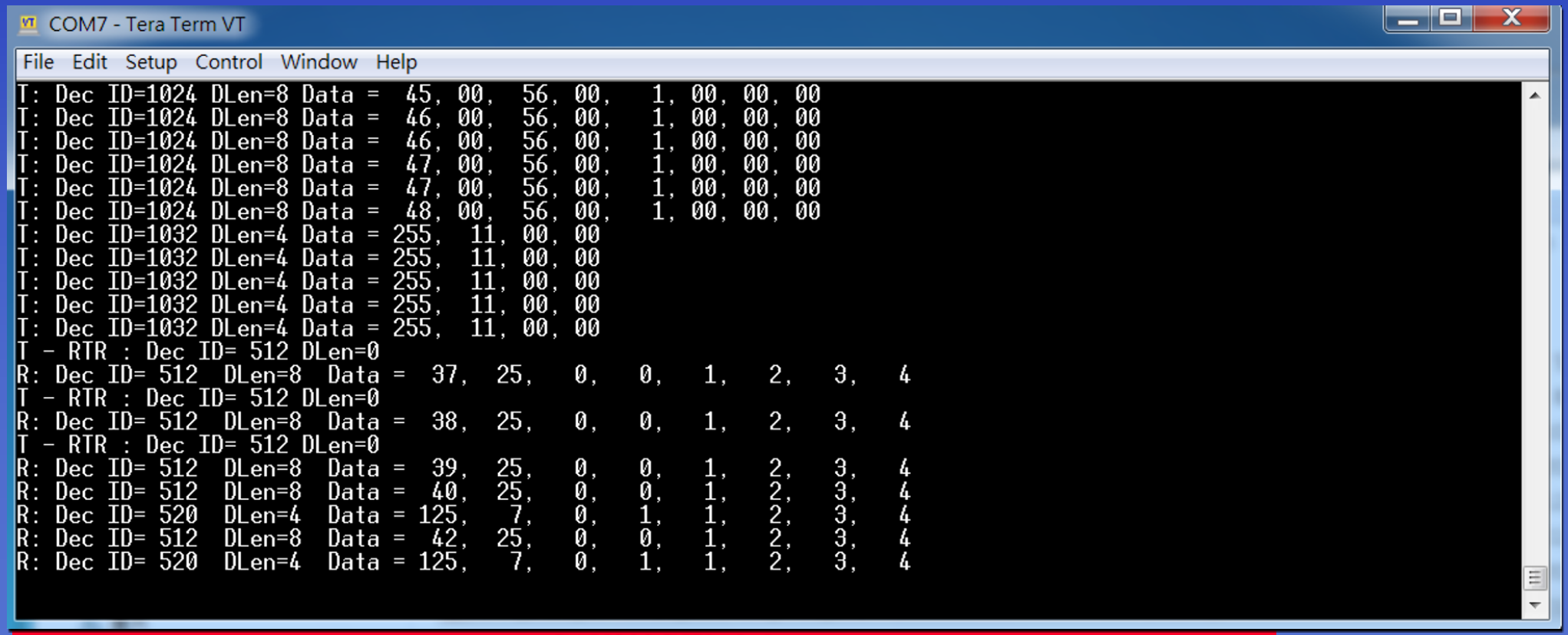
EDU1003 (APP044) 簡易的 CAN Bus Debugger

Exercise 0 – 練習使用 EDU1003(APP044)

- 將簡易的 CAN Bus Debugger 程式已燒錄工具 Program 至 APP044
  - C:\RTC\CAN202C_V2\CAN202C_Ex0_Monitor
- 安裝並執行 Tera Term 或其他終端機軟體
- 確認 USB CDC 裝置的驅動程式安裝正確
- 連接 CAN Bus Debugger 並檢查其動作是否正常

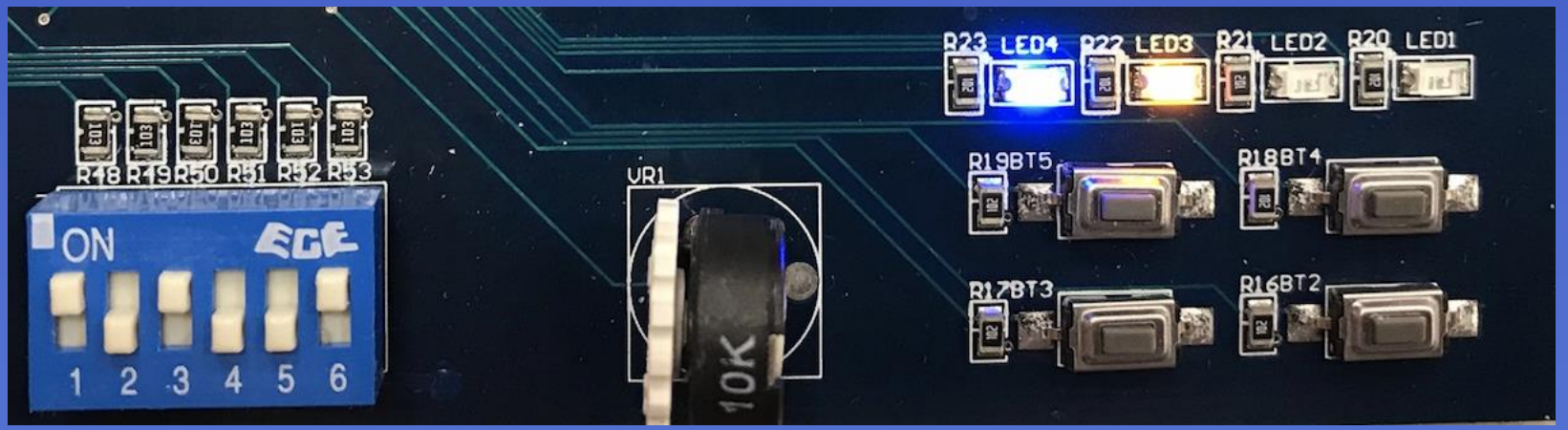
EDU1003 (APP044) 的特色

- 使用 dsPIC33EP512MU810, 有 USB & CAN
- 使用 Microchip 提供的 USB CDC 裝置，直接在終端機軟體觀察資料 (可以使用 Tera Term)
- 除觀察 CAN Bus 資料外，還有按鍵觸發功能，可送出測試 Message



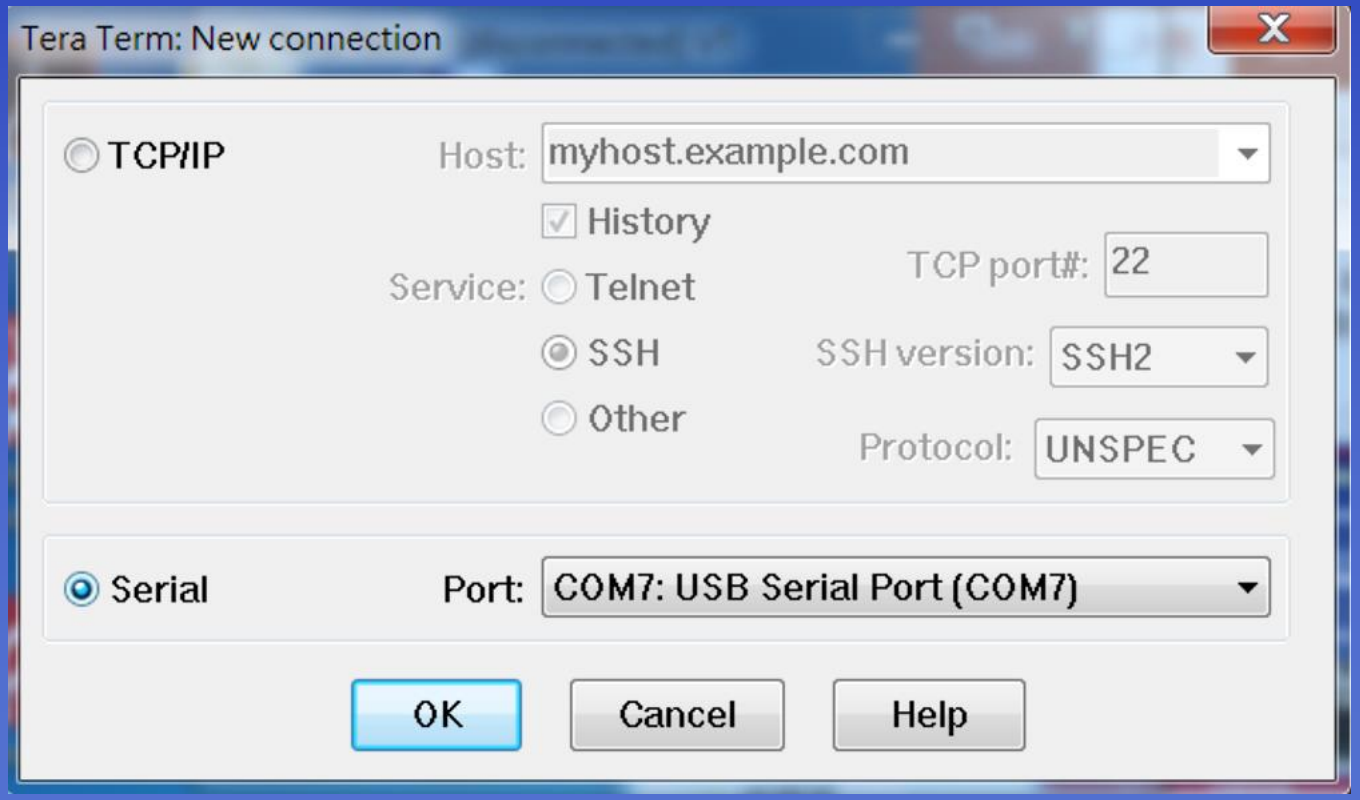
EDU1003 (APP044) 簡易的 CAN Bus Debugger

- DIP-SW6 控制以十進位或十六進位顯示
  - ON = 16 進位，OFF = 10 進位
- DIP-SW1 控制 EDU1003 送出 Message 的速度
  - ON = High Speed，OFF = Low Speed
- DIP-SW3 控制 EDU1003 的 Message ID
  - ON = 0x400 & 0x408，OFF = 0x600 & 0x608



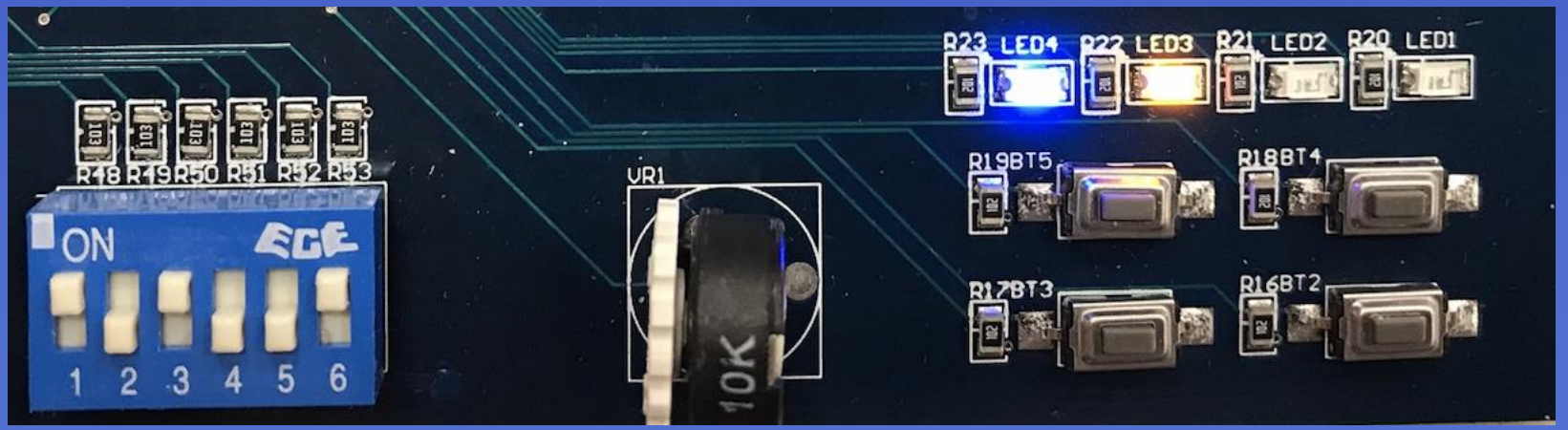
使用 Tera Term 終端機軟體

- Tera Term 的使用
  - 選取 Serial Port
  - 指向 EDU1003(APP044) 所在的 COM Port
  - 如果字體太小，可以在 Setup 功能表中修改 Font !



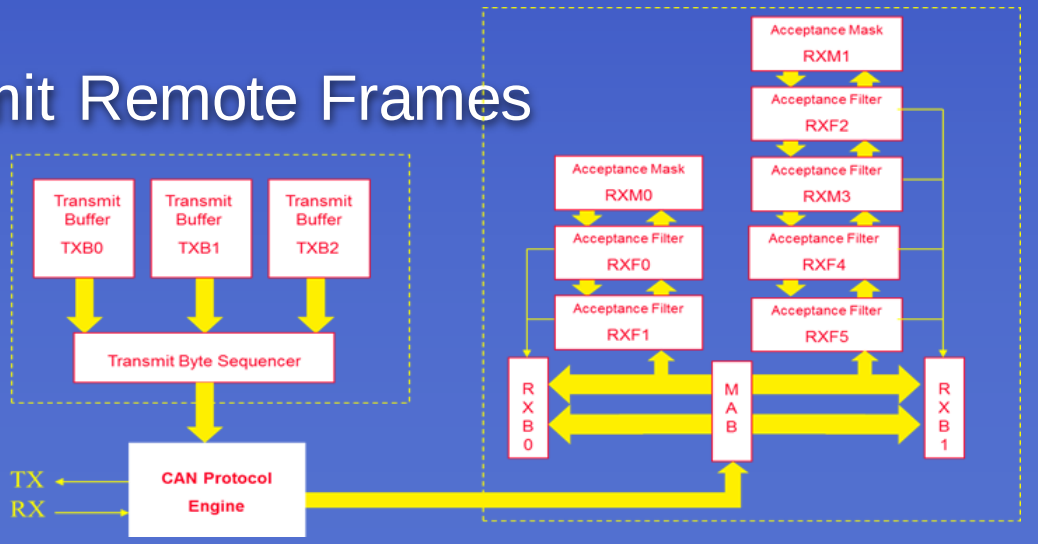
EDU1003 簡易的 CAN Bus Debugger

- 按下 BT4, 送出 EDU1003 的 CLOCK Message
  - 0x400 or 0x600
- 按下 BT2, 送出 EDU1003 的 VR1 Message
  - 0x600 or 0x608
- 按下 BT3, 送出對 Message ID = 0x200 的 RTR 請求



Standard CAN 的基本 support

- Standard CAN (Mode 0 , legacy Mode)
  - Programmable bit rate up to 1 Mbps
  - Three Transmit, 2 Receive Message Buffers
  - Six Acceptance Filters, 2 Acceptance Masks
  - Wake-up on CAN message functionality
  - Programmable time-stamp operation
  - Multiple operation modes
  - Supports to receive/transmit Remote Frames

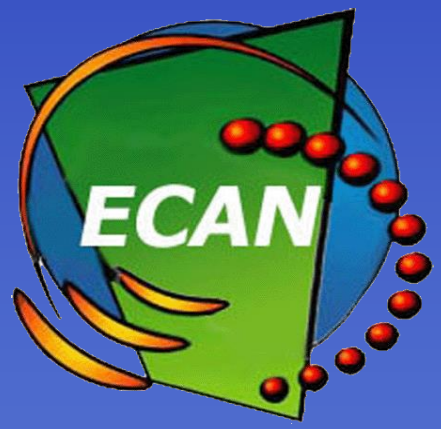


ECAN

Microchip 的 ECAN solution 概要

PICmicro®'s *ECAN* Module – PIC18F

- PIC18F 內建的 ***ECAN*** module 功能以 Mode 0 的操作模式來相容於早期的 PIC18FXX8 MCU 中的 Standard (Legacy) CAN module
- PIC18F 的 ECAN Engine 可運作於以下 3 種模式
  - "Mode 0" Legacy Mode
  - "Mode 1" Enhanced Mode
  - "Mode 2" FIFO Mode



- Modes 1 and 2 可以支援 DeviceNet 的協定

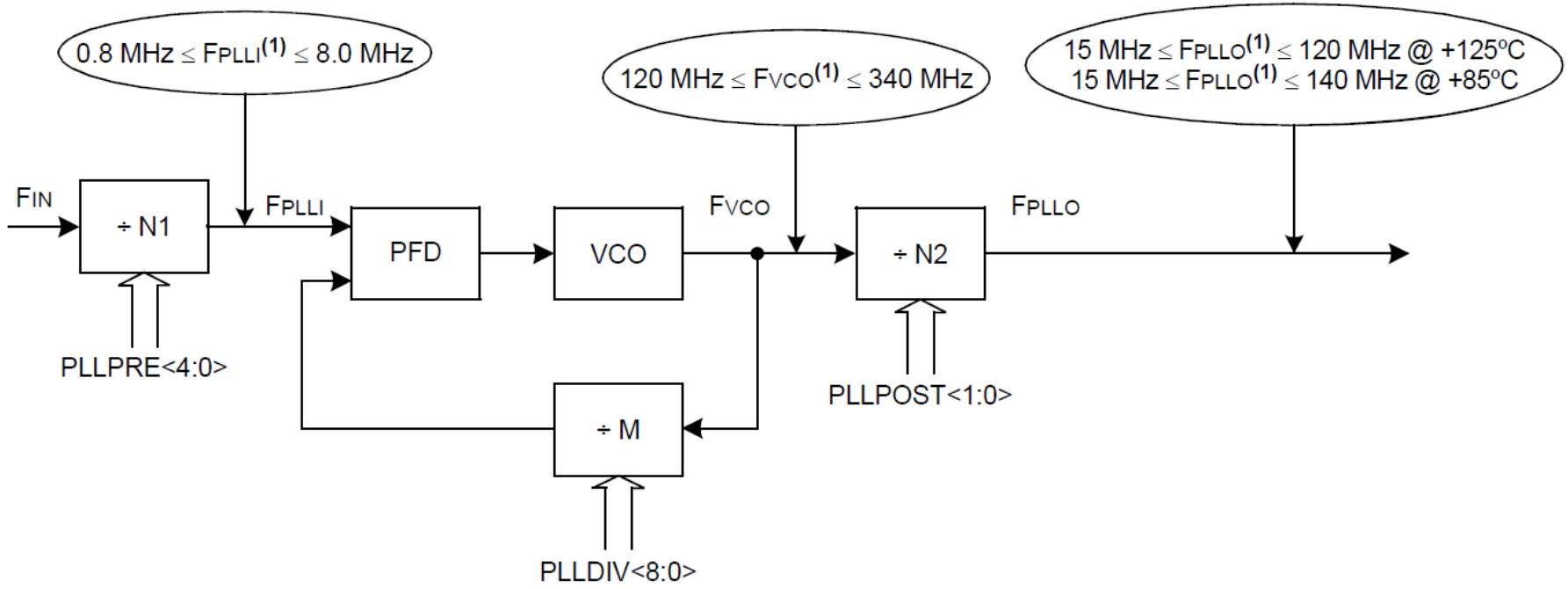
dsPIC33EP Family ECAN Engine

PICmicro®'s *ECAN* Module – dsPIC/PIC24H

- dsPIC33F 內建的 ***ECAN*** module 功能比 PIC18F 的還要優越，主要的改進都是為了要增加通信的效能以及減少程式的負擔
  - 32 個 Message Buffer
    - Buffer 0..7 可規劃為 Transmit/Receive Buffer
    - Buffer 8..31 可規劃為 Receive Buffer
    - Receive Buffer 可以用 FIFO 的方式來接收資料
  - 以 DMA 來支援 CAN 的傳送以及接收
  - 可以設定自動回覆 Remote Frame
  - 16 個 Acceptance Filter 以及 3 個 Acceptance Filter Mask
  - 支援 DeviceNet™ 要求的定址模式
  -

dsPIC33EP 系列 MCU 的主振盪器結構

Figure 7-1: dsPIC33/PIC24 Family PLL Block Diagram



Note 1: This frequency range must be met at all times.

dsPIC33EP PLL 輸出頻率的計算

- 主要的兩個暫存器 – PLLFBD & CLKDIV

```
/* Configure Oscillator to operate the device at 60Mhz
 Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
 Fosc= 8M*60/(2*2)=120Mhz for 8M input clock */
PLLFBD = 58;                        /* M=60 */
CLKDIVbits.PLLPOST = 0;             /* N1=2 */
CLKDIVbits.PLLPRE = 0;              /* N2=2 */
```

Equation 7-2: FPLLO Calculation

$$FPLLO = FIN \times \left(\frac{M}{N1 \times N2}\right) = FIN \times \left(\frac{(PLLDIV + 2)}{(PLLPRE + 2) \times 2(PLLPOST + 1)}\right)$$

Where,

$N1 = PLLPRE + 2$

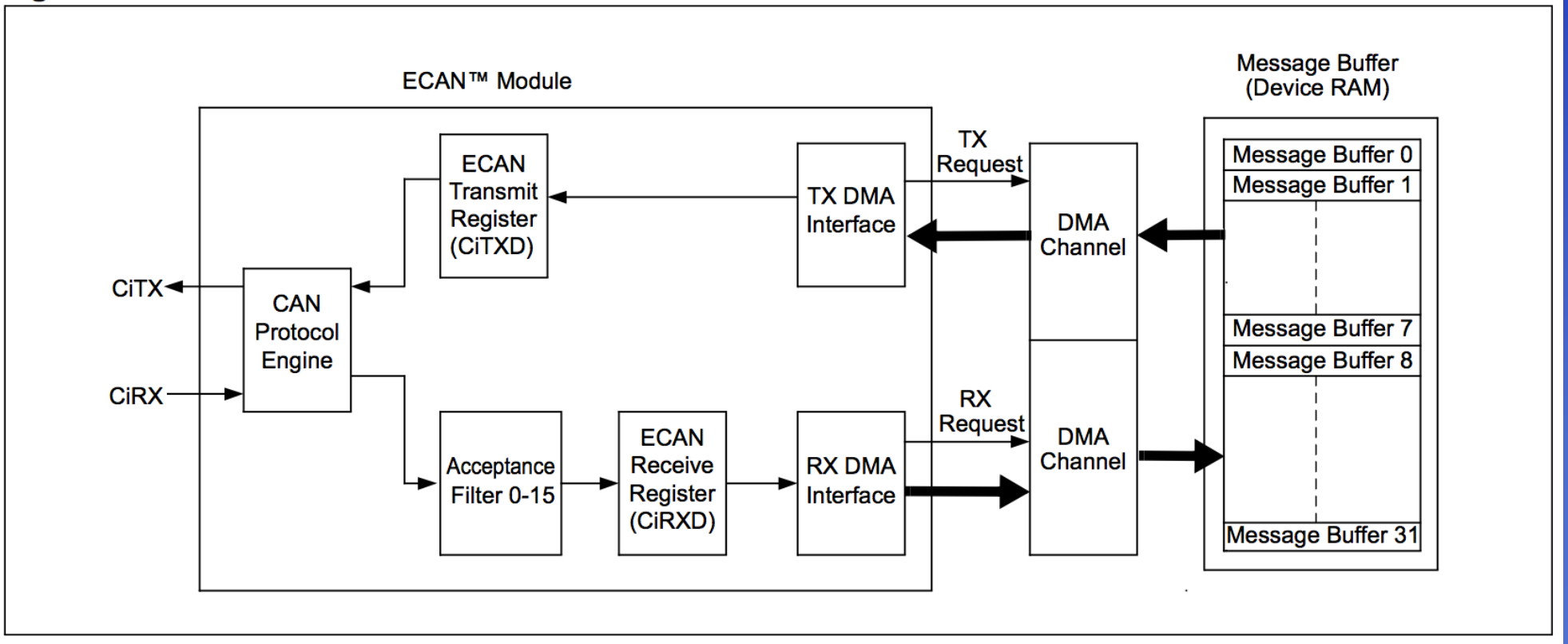
$N2 = 2 \times (PLLPOST + 1)$

$M = PLLDIV + 2$

dsPIC33EP ECAN Module & DMA

- 與 PIC18 ECAN 不同，dsPIC33EP 的 ECAN module 必須得到 DMA 的 support 才能收送資料

Figure 21-2: ECAN™ Interaction with DMA



dsPIC33EP ECAN Module & DMA

- ECAN 資料傳送
  - 被要求傳送資料的 ECAN buffer 驅動 DMA 的週期
    - 被指定做 TX 的 DMA channel 讀取觸發 DMA 傳送的 buffer 並且將資料送到 CiTXD 暫存器
- ECAN 資料接收
  - 被指定的 RX DMA channel 在進入 MAB 的資料通過過濾條件後觸發 DMA 的傳送
    - RX DMA channel 自 CiRXD 暫存器讀取資料後寫入指定的 ECAN buffer
- ECAN 資料傳送與接收需各用一個 DMA Channel

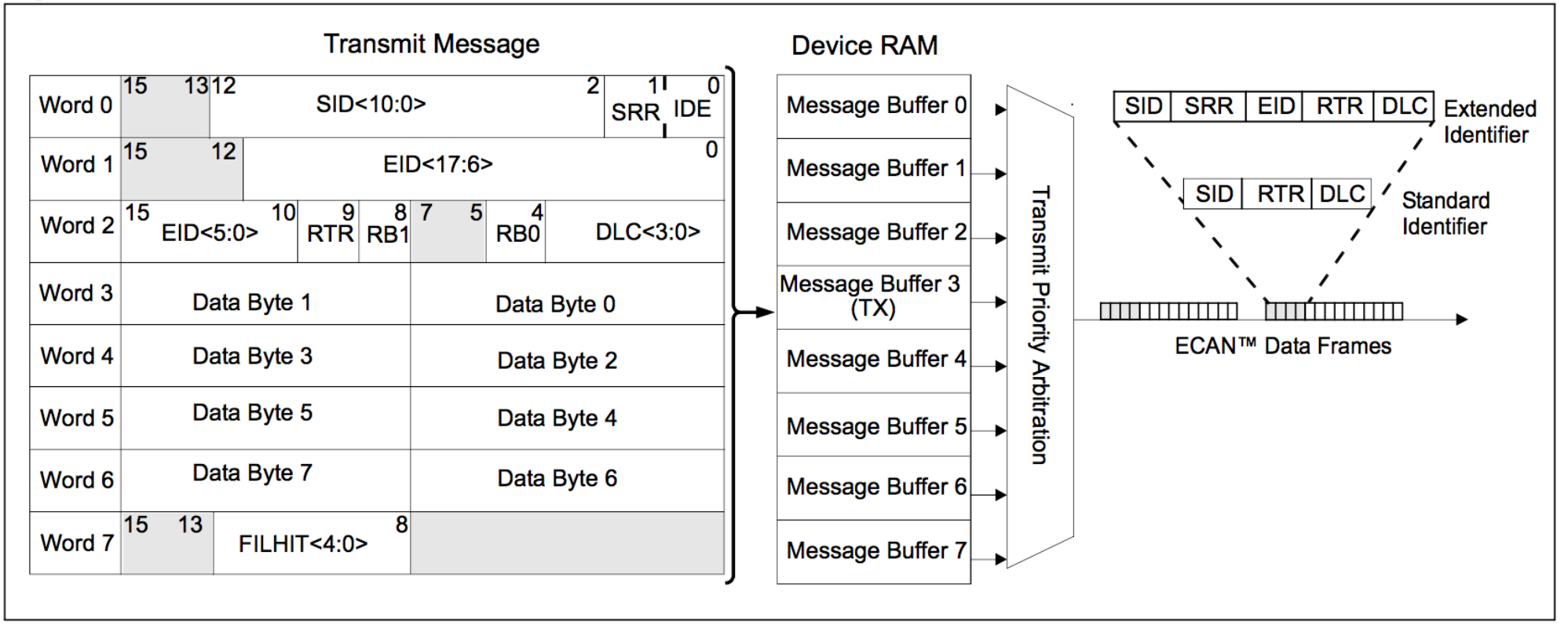
dsPIC33EP ECAN Module & DMA

- dsPIC33EP ECAN module 的 Buffer 結構如下
  - 32 個 Message Buffer
    - 32 個 Buffer 都可配規劃為 RX Buffer
    - Message Buffer 0..7 可以被規劃為 TX 或是 RX Buffer
  - 32 個 Message Buffer 可以被放置於任何的 RAM 區間，但必須是連續的區塊
    - 將 Message Buffer 放置於 Dual Port SRAM 中可以增進 DMA 資料傳輸的效率

dsPIC33EP ECAN Module – TX Buffer

- TX Buffer：最多 8 個 – 限於 Message Buffer 0..7
- 每個 TX Buffer 可獨立設定 Priority

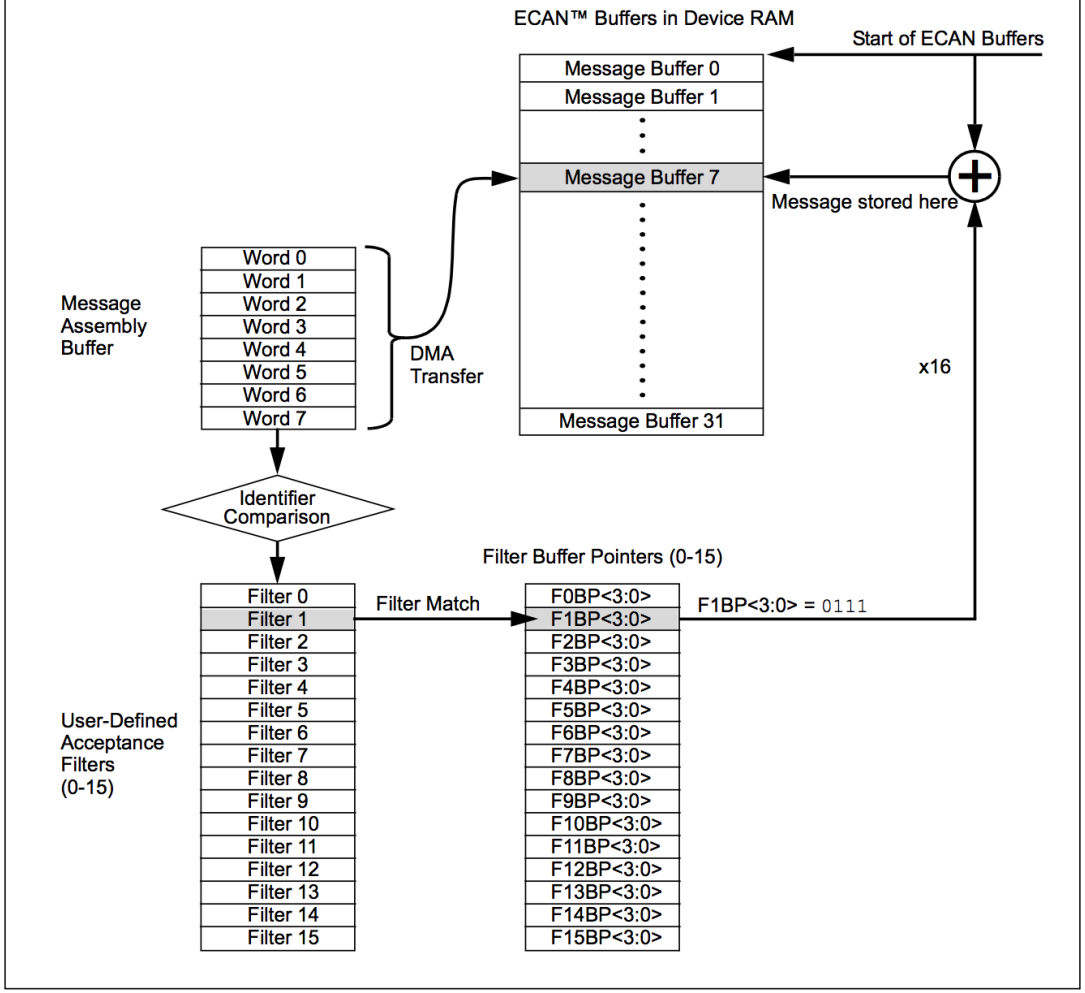
Figure 21-8: ECAN™ Transmission



dsPIC33EP ECAN Module – RX Buffer

- ECAN module 總共有 16 個 Filter
- 每一個 Filter 可以使用 FnBP 來指定當 Filter hit 時要將資料由 DMA 送至哪一個 Message Buffer
- 只有 Message Buffer 0..14 可以被 Filter 做一對一的指定
- Message Buffer 15..31 則在 FIFO Mode 時可以被使用

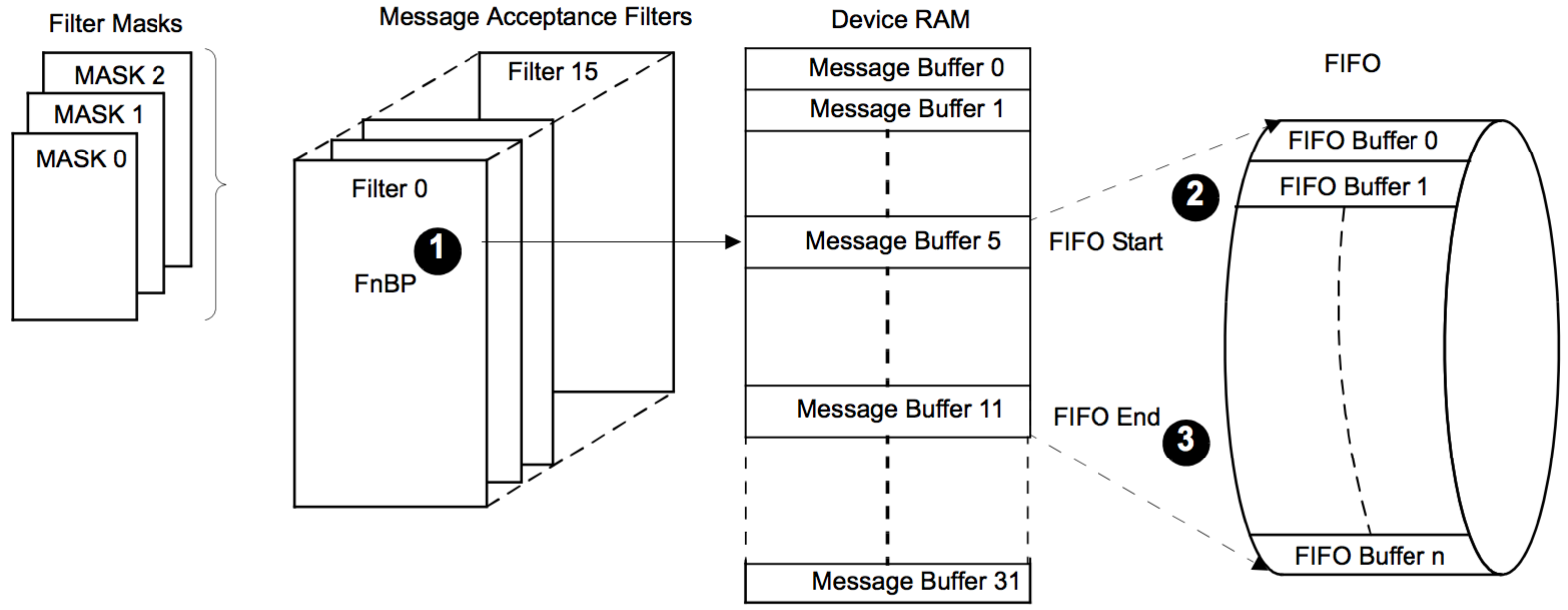
Figure 21-13: Buffer Selection and DMA Transfer



dsPIC33EP ECAN Module – RX Buffer

- 當規劃為 FIFO mode 時，最多可以有 32 個 Receive Buffer

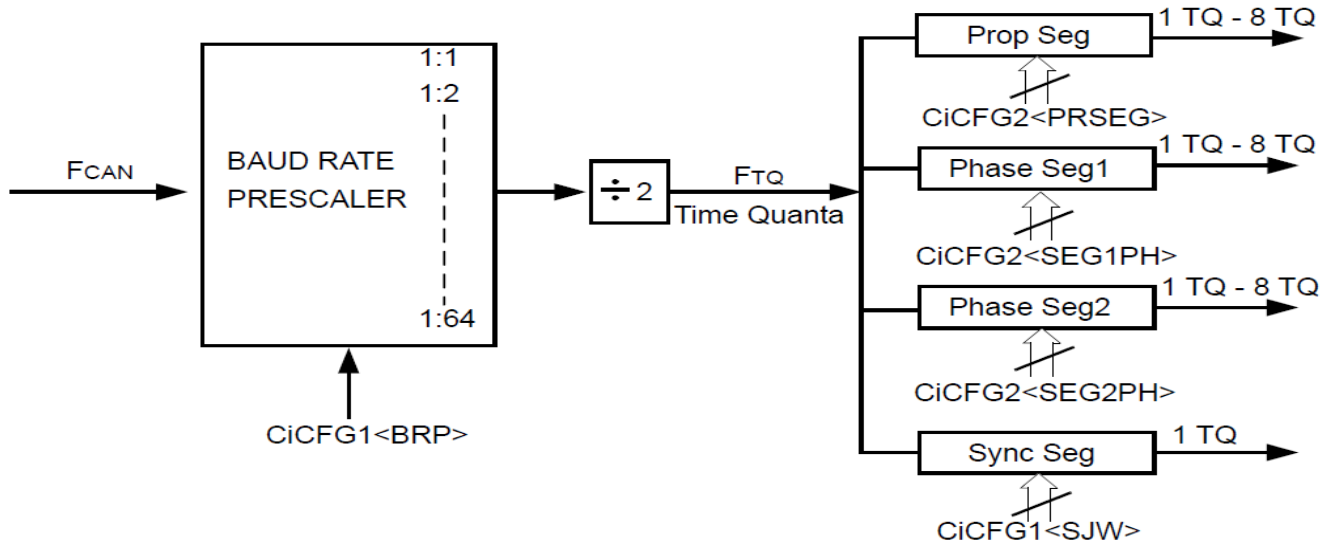
Figure 21-14: Receiving Messages in FIFO

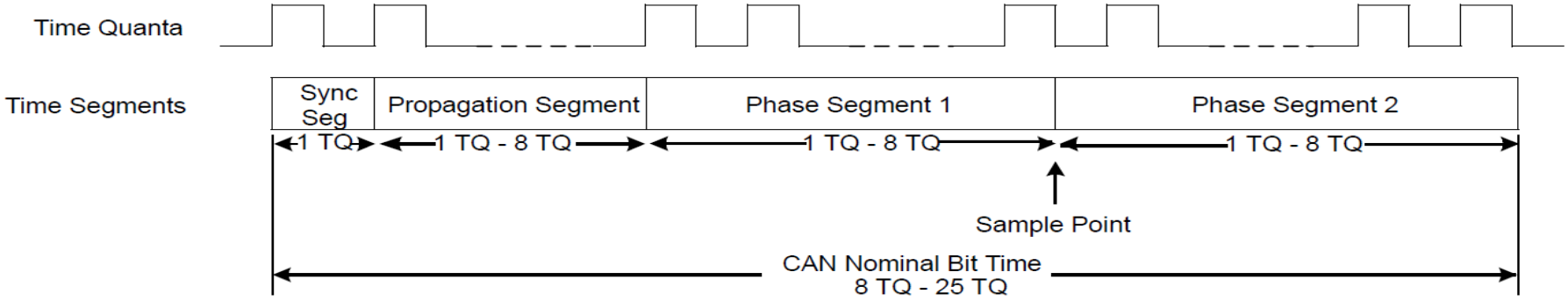


Note 1: The Acceptance Filter Buffer Pointer (FnBP) should be '1111' to store the received message in FIFO.
2: The starting address of the FIFO is specified by the FSA<4:0> bits (CiFCTRL<4:0>). In the figure, FSA <4:0> = 00101.
3: The end address of the FIFO is specified by DMABS<2:0> = 011 (CiFCTRL<15:13>).

dsPIC33EP ECAN 位元組成及時序

Figure 21-17: ECAN™ Bit Timing





dsPIC33EP ECAN Engine Clock 設定

- 兩個重要的公式如下：
  - N = 每個 bit 組成的 TQ 數量
  - FBAUD = 欲設定的 CAN 通信速率
  - FCAN 由 CiCTRL1.CANCKS 決定

Equation 21-2: Time Quantum Frequency (F_{TQ})

$$F_{TQ} = N \times F_{BAUD}$$

Equation 21-3: Baud Rate Prescaler

$$BRP<5:0> \ (CiCFG1<5:0>) = \frac{F_{CAN}}{(2 \times F_{TQ})} - 1$$

dsPIC33EP CAN Manual 對 CANCKS 的描述錯誤

- 在 CCTRL1 暫存器中對 CANCKS 的說明錯誤

Register 21-23: CiCTRL1: ECAN Control Register 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0
—	—	CSIDL	ABAT	CANCKS	REQOP<2:0>		
bit 15							bit 8

R-1	R-0	R-0	U-0	R/W-0	U-0	U-0	R/W-0
OPMODE<2:0>			—	CANCAP	—	—	WIN
bit 7							bit 0

Legend:	r = Reserved		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

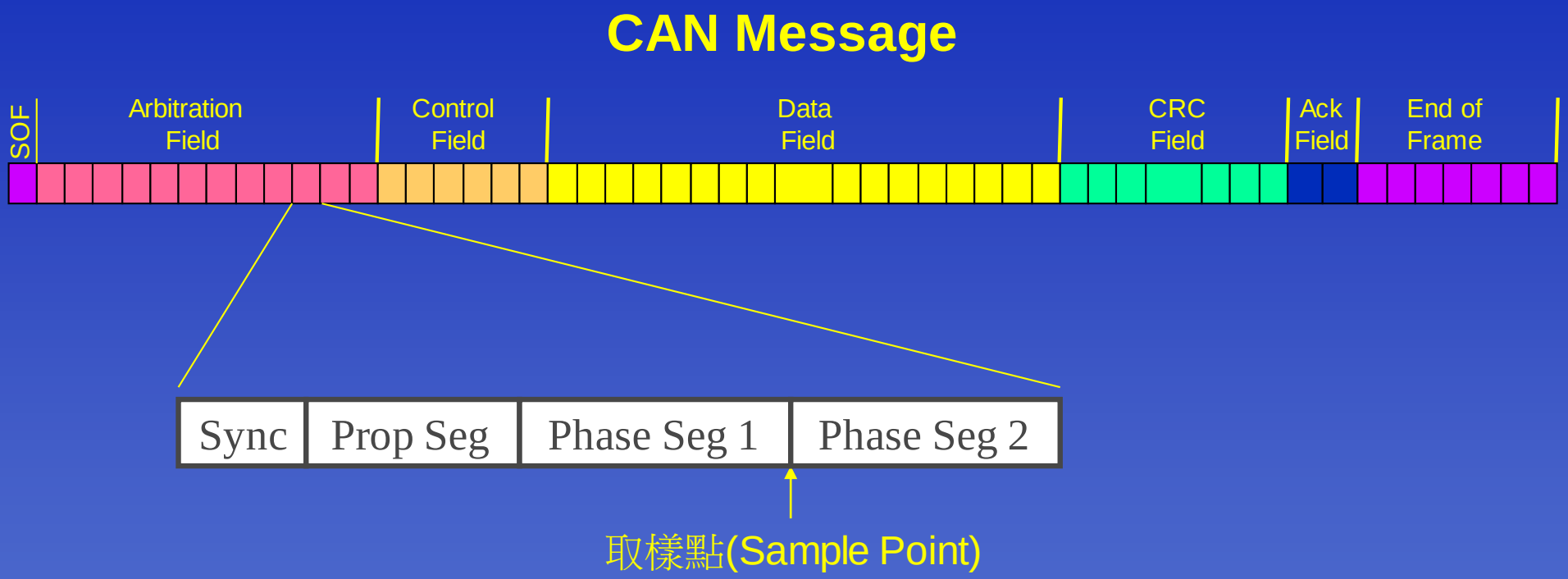
bit 15-14 **Unimplemented:** Read as '0'

bit 13 **CSIDL:** Stop in Idle Mode bit
1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation in Idle mode

bit 12 **ABAT:** Abort All Pending Transmissions bit
1 = Signal all transmit buffers to abort transmission
0 = Module will clear this bit when all transmissions are aborted

bit 11 **CANCKS:** ECAN Module Clock (FCAN) Source Select bit
1 = FCAN is equal to 2 * FP
0 = FCAN is equal to FP

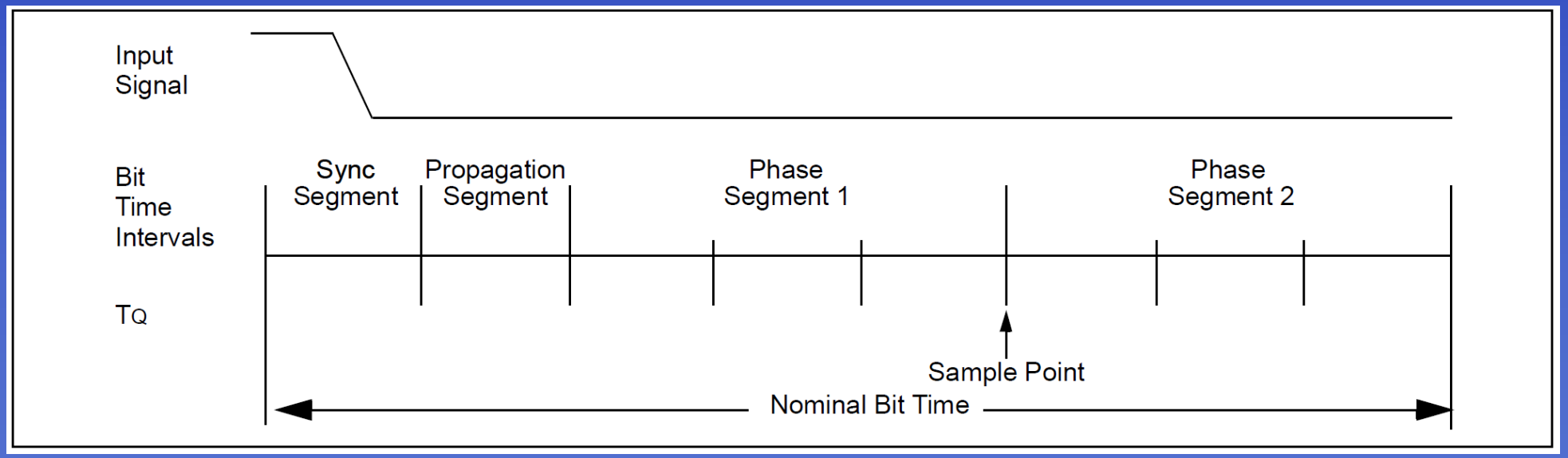
CAN Message 的位元結構



每個 CAN 的 message bit 是由 4 個 Segment 所組成

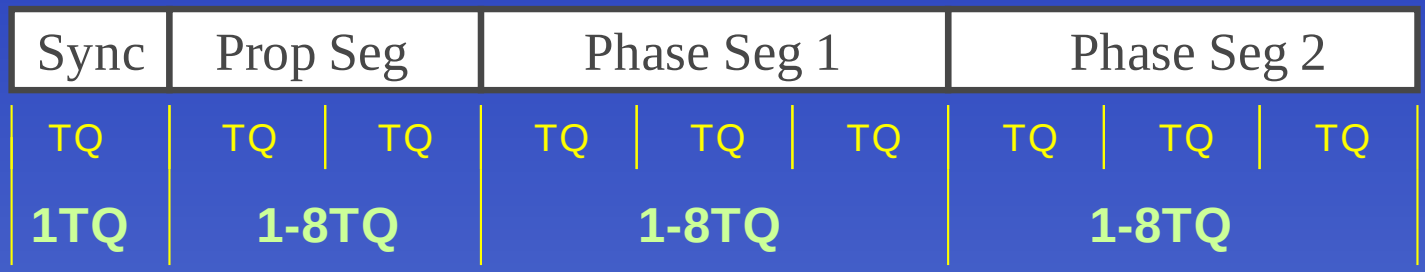
CAN Message 的位元結構

- 每一個 CAN 位元，由 4 個定義好的單元組成
- CAN 對 Bus 信號的取樣點被設定在 Phase Segment 1 與 Phase Segment 2 的交界



CAN Message 的位元結構

- 每一個 Bit Timing Segment 是由整數個單位時間所組成, 稱為 Time Quanta (TQ)

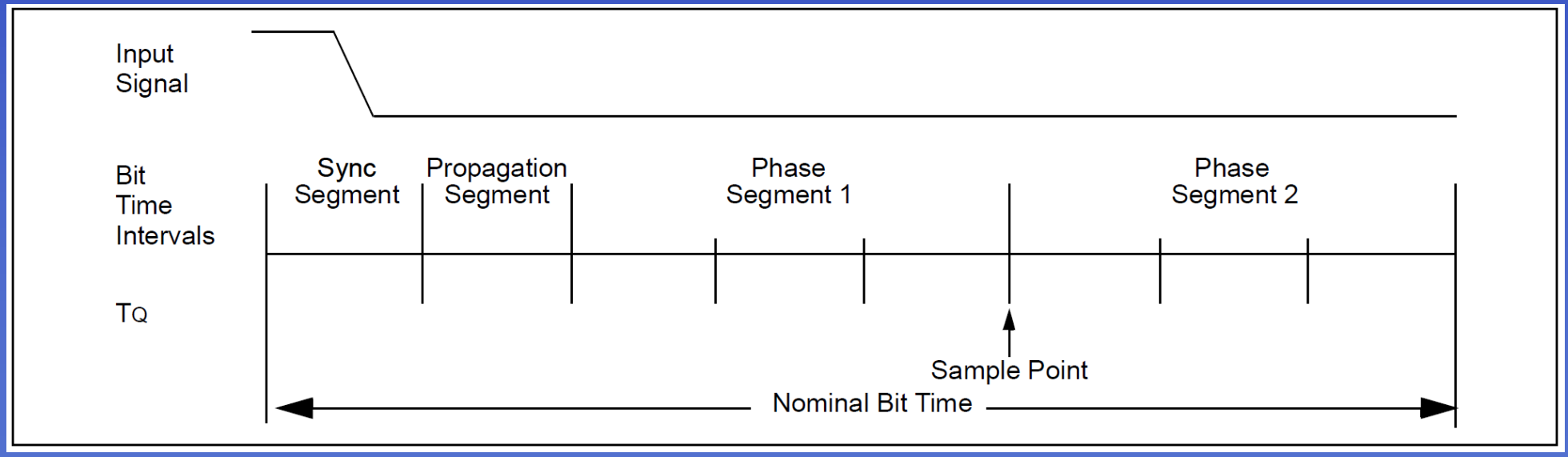


每個 Timing Segment 可被規劃為由指定數量的 TQ 所組成

- TQ 被硬體設定為 2*(BRP + 1) (Tosc) -> (Tosc = 1/Fosc)
- 更改 Baud Rate 的預除器 (BRP) 將變更 TQ 的時間
  - Min = 1:1, Max = 1:64

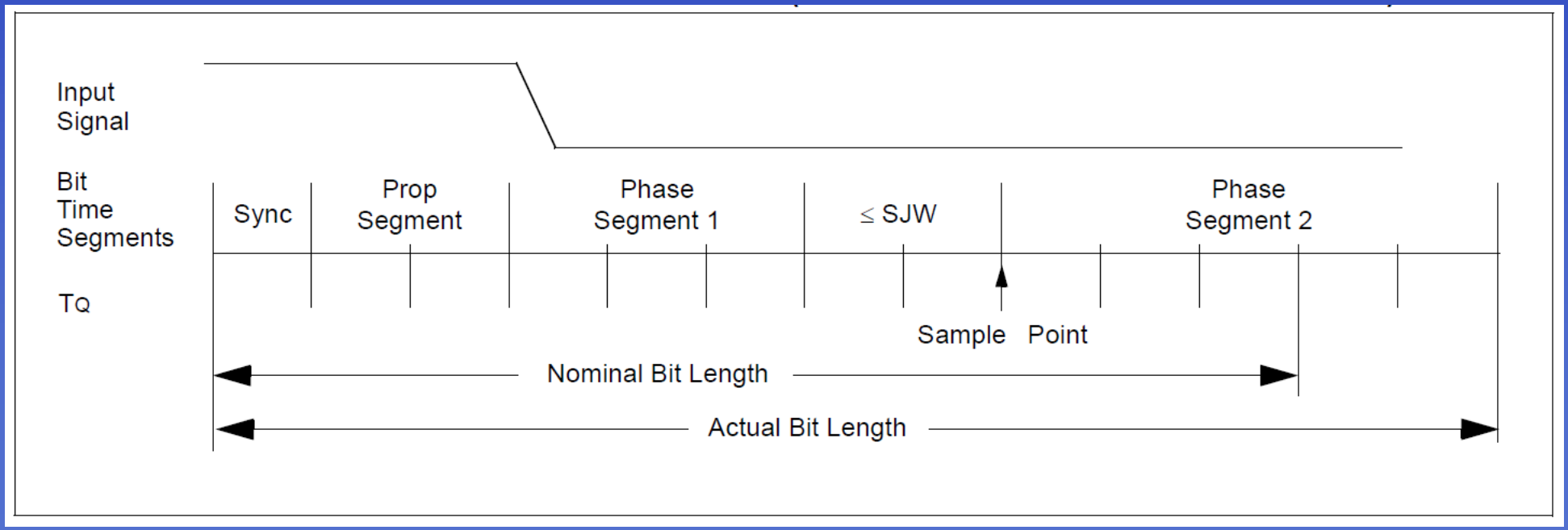
SJW 與位元同步的關係 (1)

- 理想的 CAN 資料傳輸如下，輸入訊號剛好落在 Sync Segment 中 (固定為 1 TQ)
- 但是，傳輸線越長，則信號延遲就越多！而且 CAN Engine 彼此的 clock 也有不同的誤差！
- 因為 CAN 沒有傳遞 clock 信號給對方，所以雙方只能藉由信號的 Edge 來同步
- 每個 Message 的 SOF 位元即為 CAN Bus 的硬體同步
- SOF 之後即使用信號的 Edge + SJW 來做同步 (所以 CAN 有位元填充的機制)



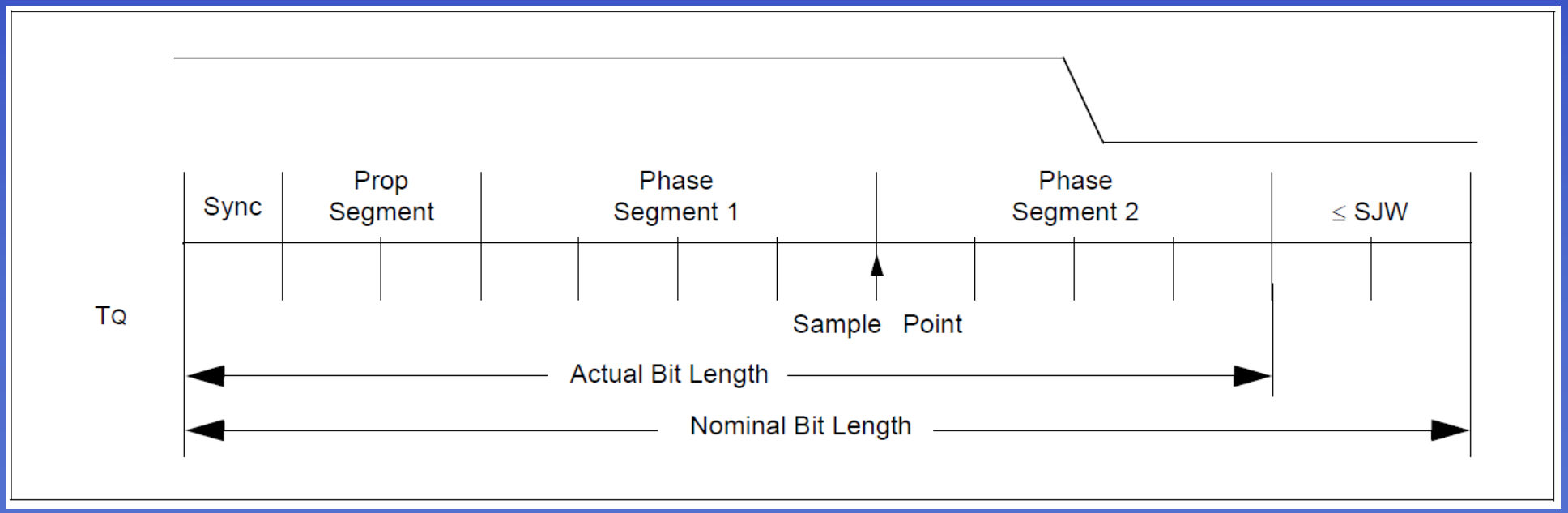
SJW 與位元同步的關係(2)

- 當信號的負緣落到了 Phase Segment 1，代表輸入信號落後了這個 CAN node 的 Timing
  - 此時使用 SJW 來延長 Phase Segment 1
  - 延長時間的最大值為 SJW 設定的時間 (最多4TQ)



SJW 與位元同步的關係(3)

- 當信號的負緣落到了 Phase Segment 2，代表輸入信號超前了這個 CAN node 的 Timing
  - 此時使用 SJW 來縮短 Phase Segment 2
  - 能縮短時間的最大值為 SJW 設定的時間 (最多4TQ)
  - 這也是為何 Phase Segment 2 的時間要 >= SJW 的主因



ECAN Module Initialization

計算各 Bit Rate Register 的設定值

- 計算條件與公式：
  - 需要的 baud rate 為 B
  - 進入 CAN Engine 的頻率為 F_{CAN}, 週期為 T_{CAN}
    - 每一個 bit 所站時間 = Tbit = 1 / B
    - TQ = 2 * (BRP + 1) * T_{CAN} (公式 1)
    - TQ = Tbit / N, N 是每一個 bit 包含的 TQ 數量 (公式 2)
    - BRP = ((F_{CAN}/(2 * N* B)) - 1 (公式 2 代入 公式 1)
    - 在 8 <= N <= 25 的條件下，選擇一個最大可能的 N，並且滿足讓計算後的 BRP 值為整數
    - 將 BRP 寫入 CiCFG1
      - Phase Segment 2 = SJW = N * 0.30 (取整數)
        - SJW 的最大值為 4 TQ
      - Prop Segment = 4 (依據傳輸線的長度可能需要調整)
      - Phase Segment1 = N - 1 - 4 - Phase Segment 2

ECAN Module Initialization

計算各 Bit Rate Register 的設定值-範例

- 在 CPU 執行頻率為 120 Mhz 的條件下規畫出 125 kbps 的 CAN Bit Rate (設定 $F_{CAN} = F_P$=60Mhz)
  - Tbit = 1 / 125 kbps = 8 uS
  - 選取 N = 20,
    - (Fosc/N) / (2 * B) = (60MHz/20)/(2 *125kbps) = 12
  - BRP = 12 - 1 = 11 (剛好整數)
  - Phase Segment 2 = SJW = 20 * 0.30 = 6
    - SJW 的最大值為 4
  - Prop Segment = 5
  - Phase Segment 1 = 20 - 1 - 5 - 6 = 8
- 最後寫入 CiCFG1 & CiCFG2 控制暫存器的結果:
  - BRP<5:0> = 11, SEG1PH = 7, SEG2PH = 5, SJW = 3, PRSEG = 4

Ecan1_config.h 中對 CAN Timing 的設定

- 在範例程式中，dsPIC33EP 被規劃為 120Mhz 頻率輸入，輸入周邊的頻率為輸入 CPU 的 ½
- 預計的 Bit Rate 為 125K
- 設定每一個 bit 的 TQ 數量為 20 (最大為25)
  - 驗算出的 BRP-VAL 為整數的話就可以被利用

```
/* CAN Baud Rate Configuration       */
#define FCAN    60000000L
#define BITRATE 125000
#define NTQ     20  // 20 Time Quanta in a Bit Time
#define BRP_VAL ( (FCAN / (2 * NTQ * BITRATE)) - 1 )
```

Ecan1_config.c 對 CAN Timing 的規劃

```
/* Synchronization Jump Width set to 4 TQ */
C1CFG1bits.SJW = 0x3;

/* Baud Rate Prescaler */
C1CFG1bits.BRP = BRP_VAL;

/* Phase Segment 1 time is 8 TQ */
C1CFG2bits.SEG1PH = 0x7;

/* Phase Segment 2 time is set to be programmable */
C1CFG2bits.SEG2PHTS = 0x1;

/* Phase Segment 2 time is 6 TQ */
C1CFG2bits.SEG2PH = 0x5;

/* Propagation Segment time is 5 TQ */
C1CFG2bits.PRSEG = 0x4;

/* Bus line is sampled three times at the sample point */
C1CFG2bits.SAM = 0x1;
```

dsPIC33EP ECAN Transmit 設定

- ECAN Transmit 設定的必要程序如下
  - 完成 ECAN Message Buffer 與 DMA channel 的連結
  - 指定要做 CAN Transmit 的 Message Buffer
  - 設定被指定 Transmit Message Buffer 的 Priority
  - 將欲送出的資料寫入該指定的 Message Buffer
    - Identifier 、Data Length, ID Type, Data
  - 設定 Transmit Request 位元來要求傳送
    - CiTRmnCON.TXREQm or CiTRmnCON.TXREQn
  - CAN TX Interrupt 可以被 Enable or Disable

設定 dsPIC33EP DMA

- 以 DMA channel 為例，需要設定的暫存器如下：
  - DMA0CON
    - DMA 的 Mode
      - Word or Byte
      - Direction
      - Address Mode
      - Operation Mode
  - DMA0PAD
    - DMA 連結的暫存器之位址
  - DMA0CNT
    - DMA 資料傳輸的長度
  - DMA0REQ
    - DMA 連結的周邊中斷號碼
  - DMA0STAL
    - DMA Buffer 起始位址 Low
  - DMA0STAH
    - DMA Buffer 起始位址 High

```
void DMA0Init( void )
{

    DMA0CON = 0x2020;
    DMA0PAD = ( int ) &C1TXD;   /* ECAN 1 (C1TXD) */
    DMA0CNT = 0x0007;
    DMA0REQ = 0x0046;           /* ECAN 1 Transmit */

    #ifdef _HAS_DMA_
    DMA0STAL = __builtin_dmaoffset( ecan1msgBuf );
    DMA0STAH = __builtin_dmapage( ecan1msgBuf );
    #else
    DMA0STAL = (uint16_t)(int_least24_t)(&ecan1msgBuf);
    DMA0STAH = 0;
    #endif
    DMA0CONbits.CHEN = 1;
}
```

dsPIC33EP ECAN Message Buffer 宣告

- 在 XC16 中，宣告一個大小為 32 * 8 words 的自定資料型別 ECAN1MSGBUF (每個 Message Buffer 的大小為 8 words)
- 利用此資料型別宣告一名稱為 can1msgBuf 的 Buffer，並對齊 32 * 16 bytes 的區塊，確定這個 Buffer 有足夠的連續空間
- DMA 的起始位址在程式中會被指到這一個 buffer 的開頭位址

```
#define ECAN1_MSG_BUF_LENGTH    32
typedef uint16_t  ECAN1MSGBUF[ECAN1_MSG_BUF_LENGTH][8];

// Define ECAN Message Buffers
__eds__ ECAN1MSGBUF ecan1msgBuf __attribute__( (space(eds), aligned(ECAN1_MSG_BUF_LENGTH * 16)) );
```

dsPIC33EP ECAN 的 CiTRmnCON

- 每個控制暫存器包含兩個 Message Buffer 的控制位元
  - TXEN : 1 = 設定此 Message Buffer 為 Transmit Buffer
  - TXREQ : 1 = 傳送要求位元
  - RTREN : 1 = Enable 自動 RTR Frame 回應
  - TXmnPRI : 設定這個 Transmit Buffer 的優先等級 (11 最高, 00 最低)
    - 如果兩個 Buffer 的 TXmnPRI 設定相同，則 Buffer 號碼較大的優先權也較高

Register 21-25: CiTRmnCON: ECAN TX/RX Buffer m Control Register (m = 0,2,4,6; n = 1,3,5,7)

R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
TXENn	TXABTn	TXLARBn	TXERRn	TXREQn	RTRENn	TXnPRI<1:0>	
bit 15							bit 8

R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
TXENm	TXABTm(1)	TXLARBm(1)	TXERRm(1)	TXREQm	RTRENm	TXmPRI<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

起始 dsPIC33EP ECAN 傳送的範例

- 在所有資料都填入 Message Buffer 後，再設定 TXREQx 位元
- 經由 DMA 將 TX Message Buffer 寫到 CTXD 的動作會自動被處發，不需要致能中斷

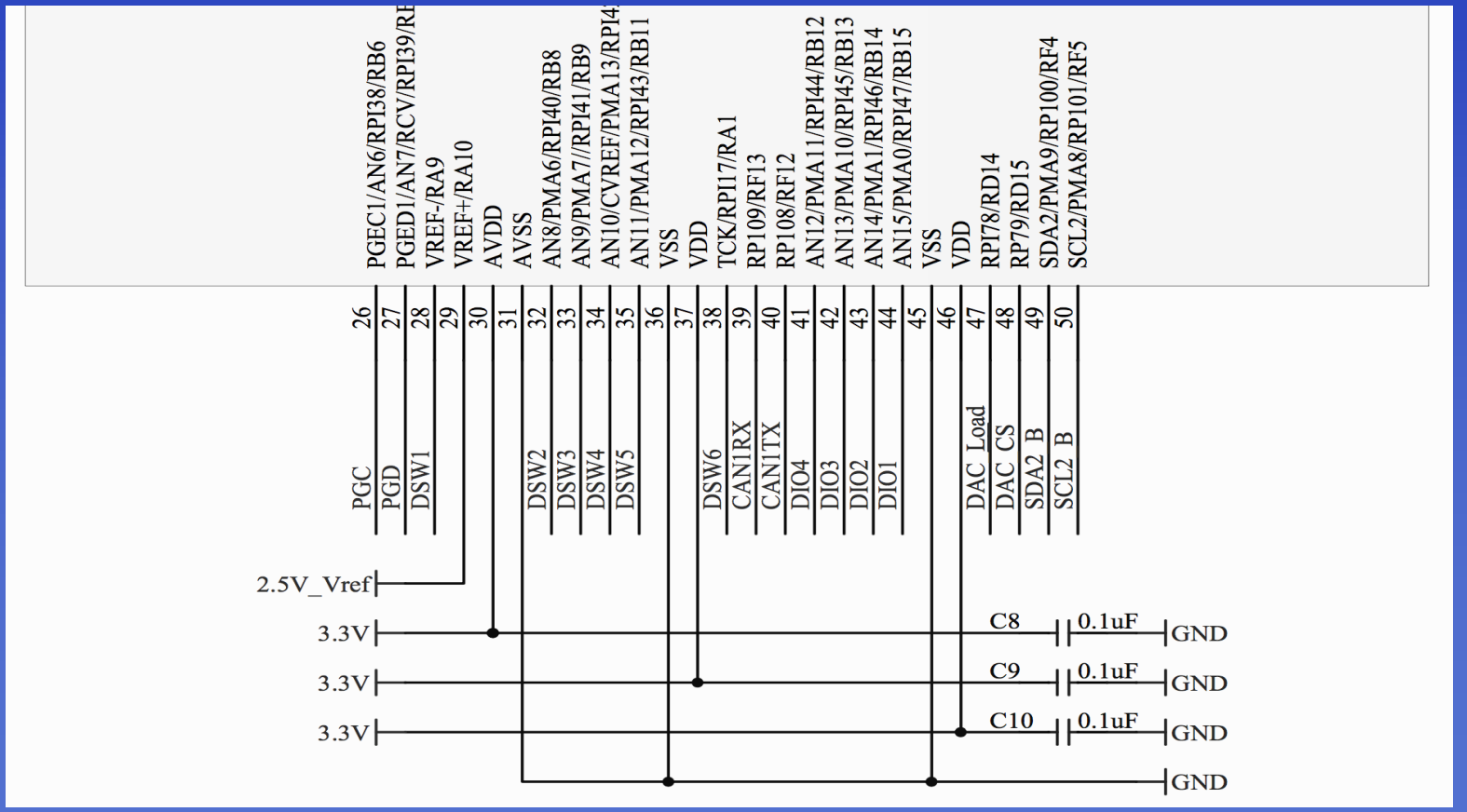
```
void    CAN_Send_CLOCK_Message(void)
{

        Ecan1WriteTxMsgBufId( 0, CLOCK_ID, 0, 0 );
        Ecan1WriteTxMsgBufData( 0, 8,  (unsigned int) CLOCK1.Second,
                                       (unsigned int) CLOCK1.Minute,
                                       (unsigned int) CLOCK1.Hour ,
                                       (unsigned int) BOARD_ID.ID_Byte);
        C1TR01CONbits.TXREQ0 = 1 ;

}
```

CANTX & CANRX 接腳的設定

- dsPIC33EP 支援 PPS (Peripheral Pin Select)
- 使用者必須參考實際電路然後對必要暫存器作設定
- RPINRxx & RPORxx 暫存器



CANTX & CANRX 接腳的設定

- 在 ecan1_config.c 中對 CAN1TX & CAN1RX 腳位的規劃 … (詳細請參考 IO_PPS 章節及 dsPIC33EP 系列資料手冊)

C1TX	001110	RPn tied to CAN1 Transmit
C2TX	001111	RPn tied to CAN2 Transmit

RPOR12	0698	—	—	RP112R<5:0>	—	—	RP109R<5:0>	0000
RPOR13	069A	—	—	RP118R<5:0>	—	—	RP113R<5:0>	0000
RPOR14	069C	—	—	RP125R<5:0>	—	—	RP120R<5:0>	0000
RPOR15	069E	—	—	RP127R<5:0>	—	—	RP126R<5:0>	0000

```
/* Setup I/O pins */
// The PPS configuration varies from device to device. Refer the datasheet of the device being used and
// use the appropriate values for the RPINR/RPOR registers.
RPINR26bits.C1RXR = 108;          //set CAN1 RX to RP108  (40)
RPOR12bits.RP109R = 14;           //RPOR12bits.RP109R = 14; set CAN1TX to RP109       (39)
```

Exercise - 1

ECAN Module 的初始設定

&

資料傳送

練習一　CAN 資料傳送

- 練習一功能
  - 規劃 dsPIC33EP 工作頻率為 120 Mhz
  - 以F_{CAN} = 60 Mhz 為依據來規劃 ECAN Module
  - 將 Message Buffer 0 設定為 TX Message Buffer
    - Ecan1Init(), Ecan1ClkInit()
  - 設定 DMA0 用來執行 CAN Message Buffer 的傳送
    - DMA0Init();
  - 使用已經寫好的 functions 來傳送資料到 CAN Bus
    - Ecan1WriteTxMsgBufId( )
    - Ecan1WriteTxMsgBufData( )

Ecan1WriteTxMsgBufId( )

- 為 Transmit Buffer 寫入 Message ID 及控制位元

```
/*********************************************************************
* Function:     void Ecan1WriteTxMsgBufId(uint16_t buf, int32_t txIdentifier, uint16_t ide,
*               uint16_t remoteTransmit)
*
* PreCondition:  None
*
* Input:        buf     -> Transmit Buffer Number
*               txIdentifier ->
*                     Extended Identifier (29-bits):
*                     Standard Identifier (11-bits) :
*
*               ide ->   "0"  Message will transmit standard identifier
*                        "1"  Message will transmit extended identifier
*
*               remoteTransmit -> "0" Message transmitted is a normal message
*                                 "1" Message transmitted is a remote message
* Overview:      This function configures ECAN1 message buffer.
*********************************************************************/
```

Ecan1WriteTxMsgBufData( )

- 為 Transmit Buffer 寫入資料

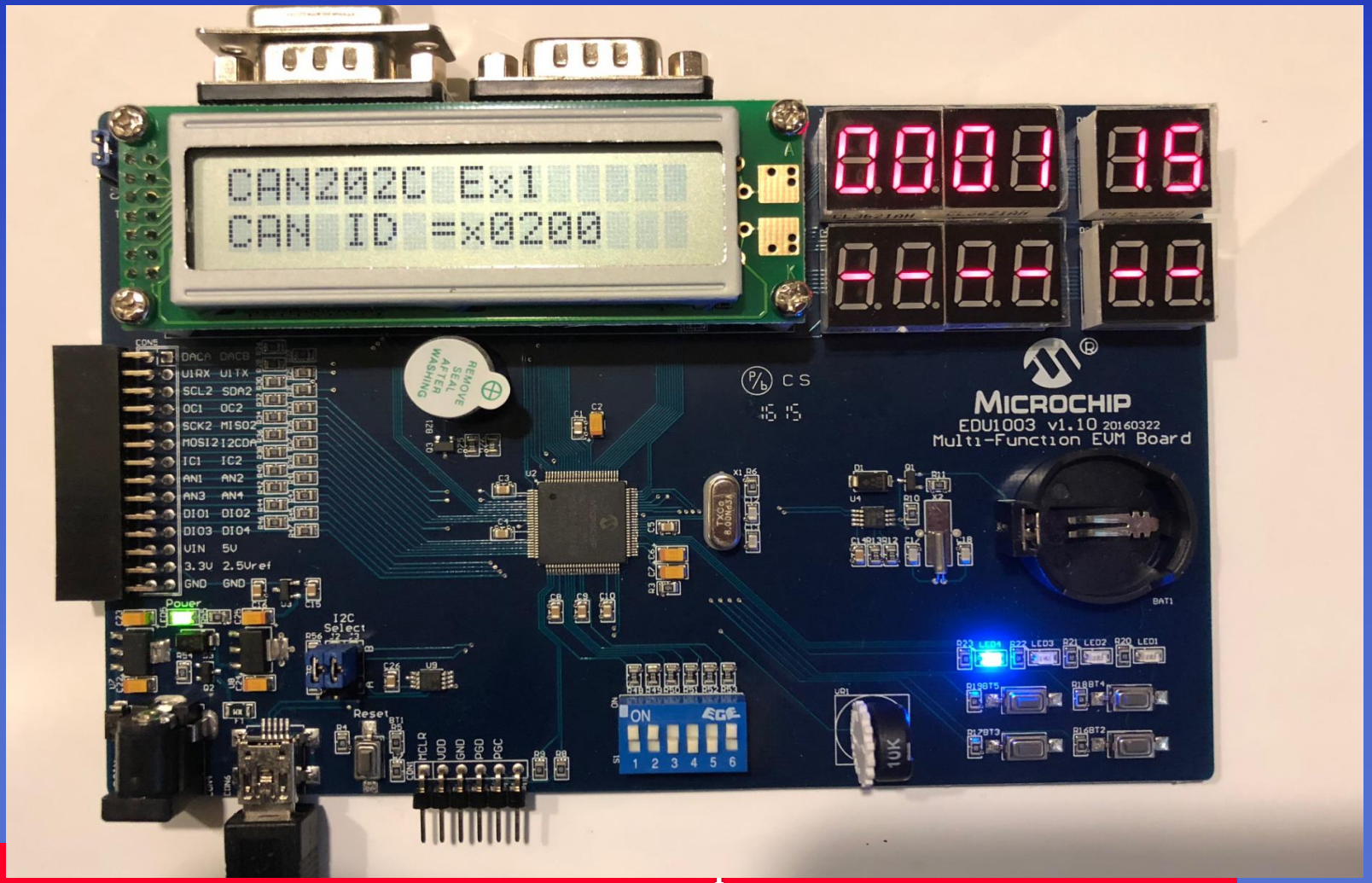
```
/*********************************************************************************
* Function:     void Ecan1WriteTxMsgBufData(uint16_t buf, uint16_t dataLength,
*    uint16_t data1, uint16_t data2, uint16_t data3, uint16_t data4)
*
* PreCondition:  None
*
* Input:          buf    -> Transmit Buffer Number
*           dataLength  -> data length in bytes.
*           actual data -> data1, data2, data3, data4
*
* Output:       None
*
* Side Effects:  None
*
* Overview:     This function transmits ECAN data.
*********************************************************************************/
```

練習一　CAN 資料傳送

- 練習一 TX Data 內容
  - MY_CANID & CLOCK_ID 都統一 define 為 0x200
  - 利用 BOARD_ID_Get() 讀取實驗板上 DIP SW 的值
    - BOARD_ID 由 BOARD_ID_Get()獲得
  - 練習一程式每隔一秒鐘會 update CLOCK1 結構中的 Second, Minute 以及 Hour
  - 利用這個一秒鐘的 update, 將 CLOCK1 結構以 Message Buffer 0 送出 (使用 CLOCK_ID 為 CAN 的　Message ID)
    - CLOCK1.Second , CLOCK1.Minutes, CLOCK1.Hour, BOARD_ID
  - 除了上述的動作之外，也一併更新第一行七段顯示器的顯示資料

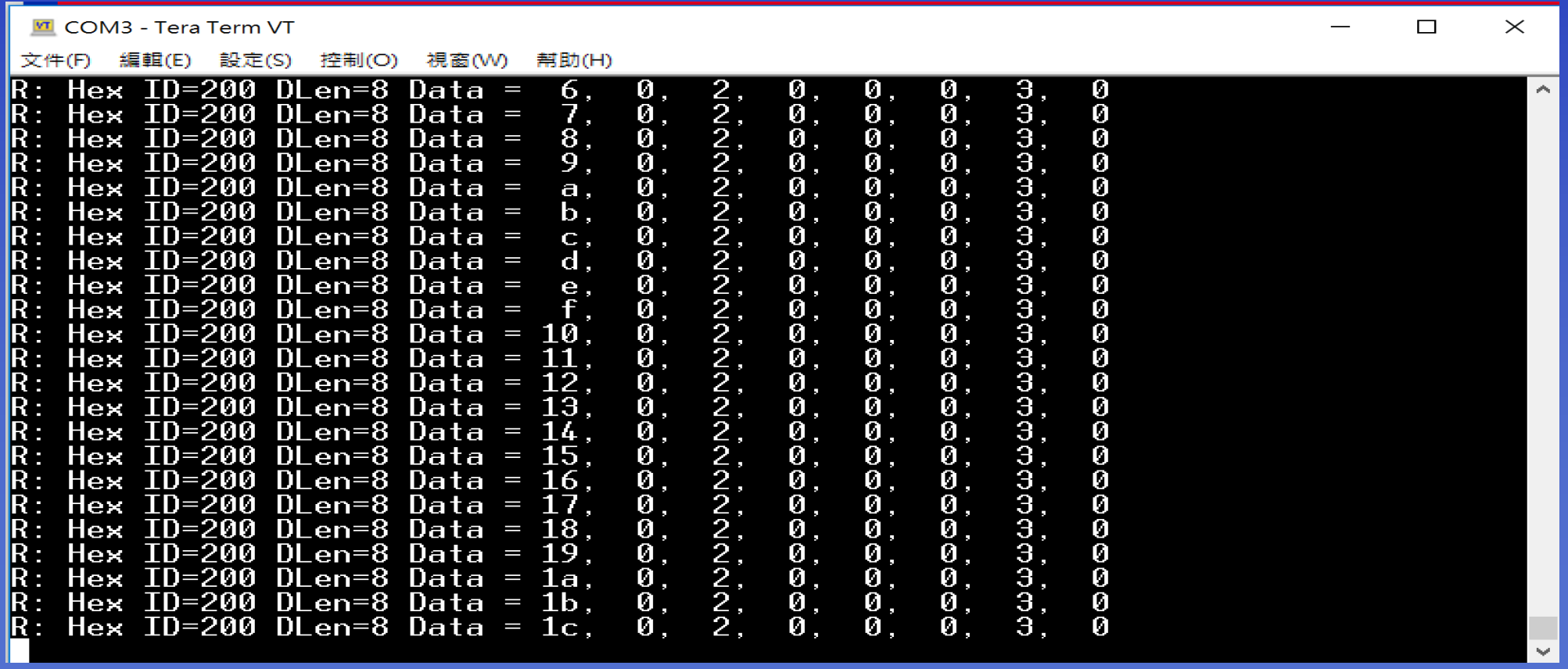
練習一　CAN 資料傳送

- 練習一執行結果如下，7-Seg Display 的第一行顯示一個每一秒 update 一次的時鐘



練習一　CAN 資料傳送

- CAN Bus Monitor 的顯示結果，可以看到資料與我們所期待的是一致的
- 注意：DIP SW 的第 6 個開關要 ON 才能做 CAN 傳送



Exercise - 2

利用兩個 Transmit Buffer
來傳送
時鐘及 ADC 的 Message

練習二

- 在程式建立一個 200ms 的 Event
- 在此 Event 中讀取 VR1 的 ADC 值
- 將此 ADC 值以 VR1_ID 為 Message ID 由 Buffer 1 送出
- 送出的長度為 4 個 Bytes (ADC 的值為 0 .. 4095)
  - 只送出 ADC 的值以及 BOARD_ID
- 如果 DIP Switch 6 = ON 的話才送出 CAN Message, 方便測試用

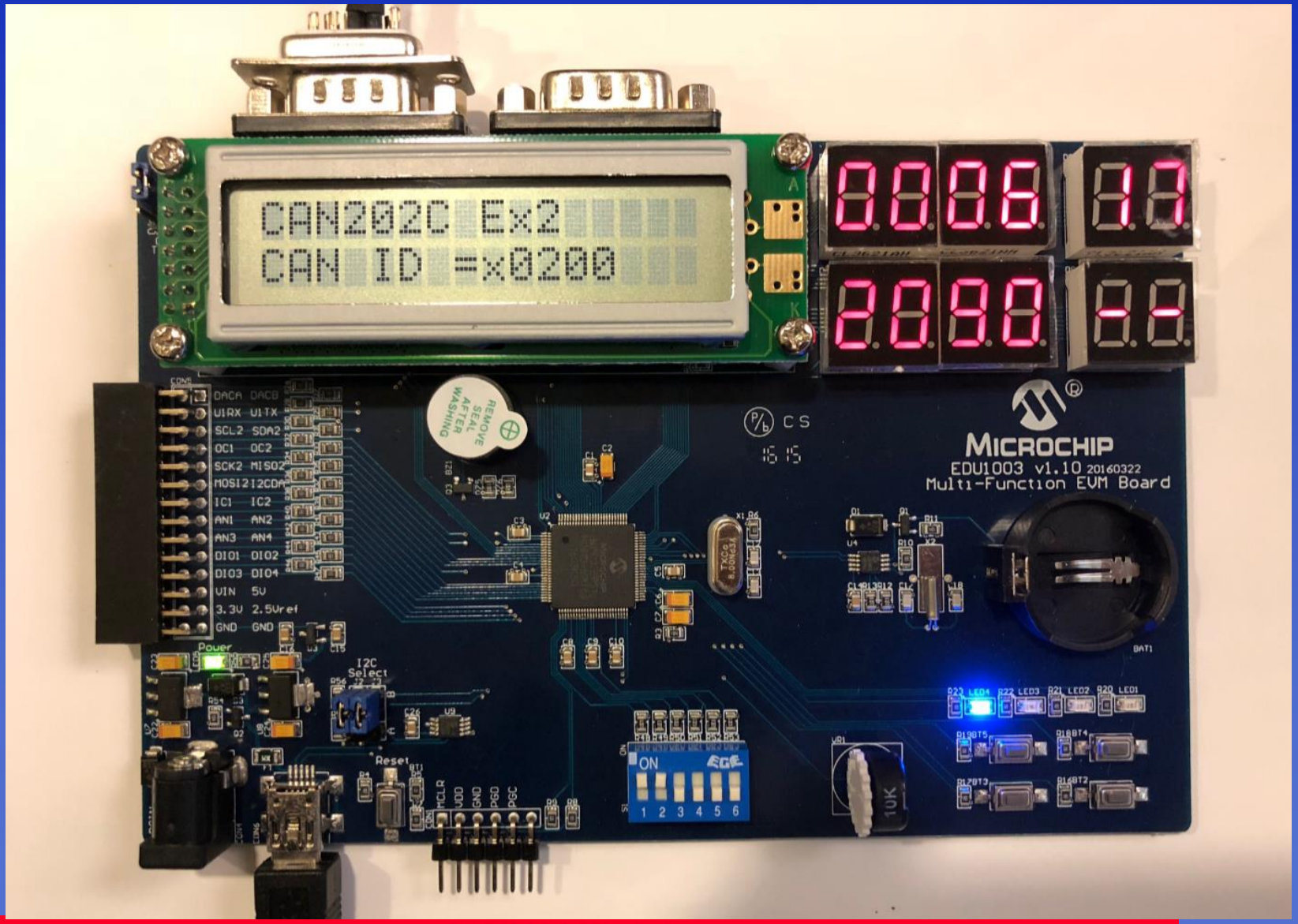
```
if (Flag_Timer1_200MS == 1 )
{
    AD1CON1bits.SAMP = 0 ;
      while ( !AD1CON1bits.DONE) ;
        VR1_Value = ADC1BUF0 ;
        ADC_Buf_to_LINE2( ) ;
            if (!PIN_DSW5)
            CAN_Send_VR1_Message( ) ;
            Flag_Timer1_200MS = 0 ;
}
```

練習二

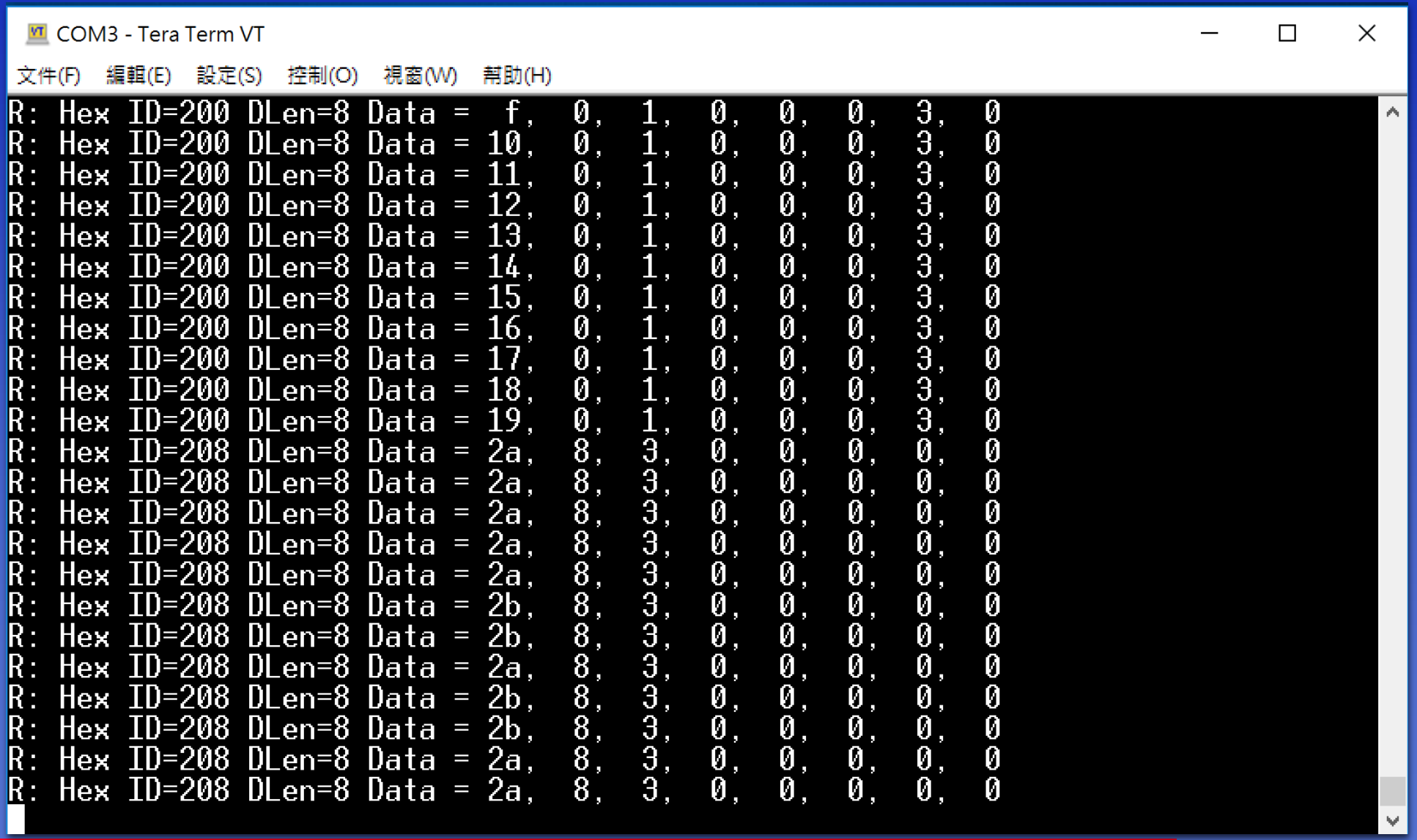
- CAN_Send_VR1_Message( ) 的內容
  - 傳送前檢查 TXREQ 位元

```
void    CAN_Send_VR1_Message(void)
{
   if ( ! C1TR01CONbits.TXREQ1  )
   {
            Ecan1WriteTxMsgBufId( 1, VR1_ID, 0, 0 );
            Ecan1WriteTxMsgBufData(    1, 4 ,   VR1_Value ,
                                       (unsigned int) BOARD_ID.ID_Byte,
                                       0x00,
                                       0x00 );
            C1TR01CONbits.TXREQ1 = 1 ;
            CANTX_FLAG.TX1_Busy = 1 ;
   }
}
```

練習二執行結果 – 學員程式

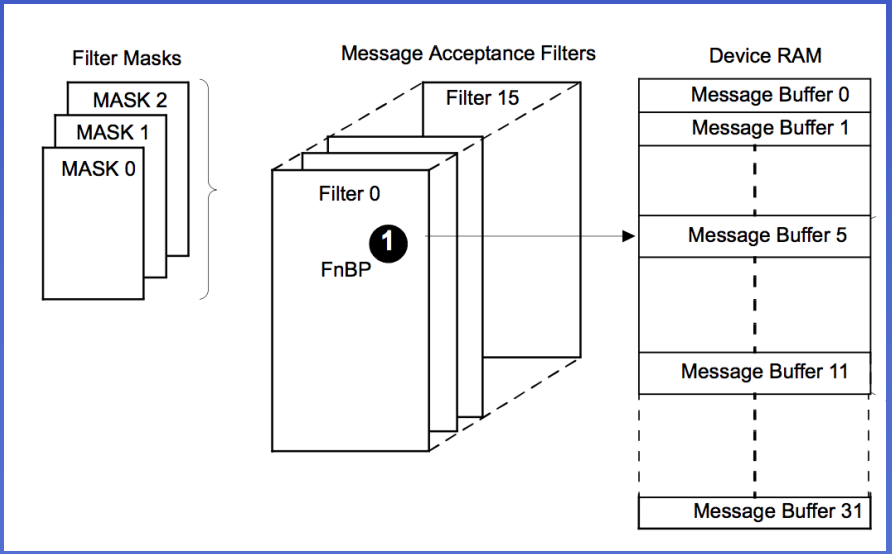


練習二執行結果 – CAN Monitor



Masks 和 Filters 的概念

- CAN 的 Message 會先被接收至 MAB ，經由 Mask & Filter 來決定在 MAB 中的 Message 是否要被接收 (以 ID – Identifier 來決定)
- Masks 決定那些 Filter 位元將被用來過濾 ID (Identifier)
  - '1' = 使用 Filter 來過濾此位元
    - 在 Identifier 中對應的位元必需與 Filter 中對應的位元相等才被接受
  - '0' = 不使用 Filter 來過濾此位元 (無條件接受)
    - 在 Identifier 中對應的位元不管為 1 或 0 都會被接
- Identifier 通過了 Mask & Filter 的篩選後該筆資料才會被正式接收



FILTER/MASK TRUTH TABLE			
Mask Bit n	Filter Bit n	Message Identifier Bit n	Accept or Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

如何為 Filter0 .. Filter15 指定 Mask

- 與 PIC18F ECAN module 不同，dsPIC33EP 的 ECAN 可以自己調整 Filter 與 Mask 的對應關係
  - CiFMSKSEL1 & CiMSKSEL2

Register 21-8: CiFMSKSEL1: ECAN Filter 7-0 Mask Selection Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F7MSK<1:0>		F6MSK<1:0>		F5MSK<1:0>		F4MSK<1:0>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F3MSK<1:0>		F2MSK<1:0>		F1MSK<1:0>		F0MSK<1:0>	
bit 7							bit 0

Register 21-9: CiFMSKSEL2: ECAN Filter 15-8 Mask Selection Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F15MSK<1:0>		F14MSK<1:0>		F13MSK<1:0>		F12MSK<1:0>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F11MSK<1:0>		F10MSK<1:0>		F9MSK<1:0>		F8MSK<1:0>	
bit 7							bit 0

如何為 Filter0 .. Filter15 指定 Message Buffer

- 使用 CiBUFPNT1 ~ CiBUFPNT4
- 每個控制暫存器控制 4 個 Filter 的 Message Buffer 對應

Register 21-10: CiBUFPNT1: ECAN Filter 0-3 Buffer Pointer Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F3BP<3:0>				F2BP<3:0>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F1BP<3:0>				F0BP<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-12 **F3BP<3:0>:** RX Buffer Mask for Filter 3 bits
1111 = Filter hits received in RX FIFO buffer
1110 = Filter hits received in RX Buffer 14
•
•
•
0001 = Filter hits received in RX Buffer 1
0000 = Filter hits received in RX Buffer 0

bit 11-8 **F2BP<3:0>:** RX Buffer mask for Filter 2 bits (same values as bits 15-12)

bit 7-4 **F1BP<3:0>:** RX Buffer mask for Filter 1 bits (same values as bits 15-12)

bit 3-0 **F0BP<3:0>:** RX Buffer mask for Filter 0 bits (same values as bits 15-12)

Exercise - 3

使用 Message Buffer 4
接收
自主機傳來的時鐘信號

練習三 – Single Message Buffer 接收

- 為了能正確地接收資料，必須完成下列工作
  - 正確地設定 Filter 的條件
    - 由 16 個 Filter 選一個來做資料的過濾
      - (例如 Filter 1)
    - 為此被選定的 Filter 指定一個 Mask
    - 為此被選定的 Filter 指定被過濾完成的資料要寫入哪一個 Buffer
- 在正確設定後，就可以在主程式或 CAN 的中斷服務程式中檢查被指定 Buffer 對應到的 RXFUL 位元
  - 確認是否有資料經由 DMA 讀入 ECAN 的 Message Buffer

練習三 – Single Message Buffer 接收

- Ecan1_config.c 中檢查 Ecan1Init( ) 副程式中是否有對 Filter 以及 Mask 做正確設定
  - Filter 1 被設定過濾 EXT_CLOCK_ID
  - Filter 1 被指定將過濾後的資料寫入 Buffer 4
  - Filter 1 被指定使用 Mask 1 來決定哪些位元要被檢查
  - Mask1 被設定為檢查每一個位元
  - EXT_CLOCK_ID 在 common.h 中被宣告

```
Ecan1WriteRxAcptFilter( 1, (0x00 | EXT_CLOCK_ID), 0, 04, 1 );
Ecan1WriteRxAcptFilter( 2, (0x00 | EXT_VR1_ID), 0, 15, 1 );

Ecan1WriteRxAcptMask( 1, 0x1FFFFFFF, 1, 0 );
```

Ecan1WriteRxAcptFilter( )

- Ecan1WriteRxAcptFilter( ) 的各參數用法如下

Ecan1WriteRxAcptFilter(int n, long identifier, unsigned int exide,unsigned int bufPnt,unsigned int maskSel)

n = 0 to 15 -> Filter number

identifier -> SID <10:0> : EID <17:0>

exide = 0 -> Match messages with standard identifier addresses
exide = 1 -> Match messages with extended identifier addresses

bufPnt = 0 to 14 -> RX Buffer 0 to 14
bufPnt = 15 -> RX FIFO Buffer

maskSel = 0 -> Acceptance Mask 0 register contains mask
maskSel = 1 -> Acceptance Mask 1 register contains mask
maskSel = 2 -> Acceptance Mask 2 register contains mask
maskSel = 3 -> No Mask Selection

Ecan1WriteRxAcptMask( )

- Ecan1WriteRxAcptMask( ) 的各參數用法如下

Ecan1WriteRxAcptMask(int m, long identifierMask, unsigned int mide, unsigned int exide)

m = 0 to 2 -> Mask Number

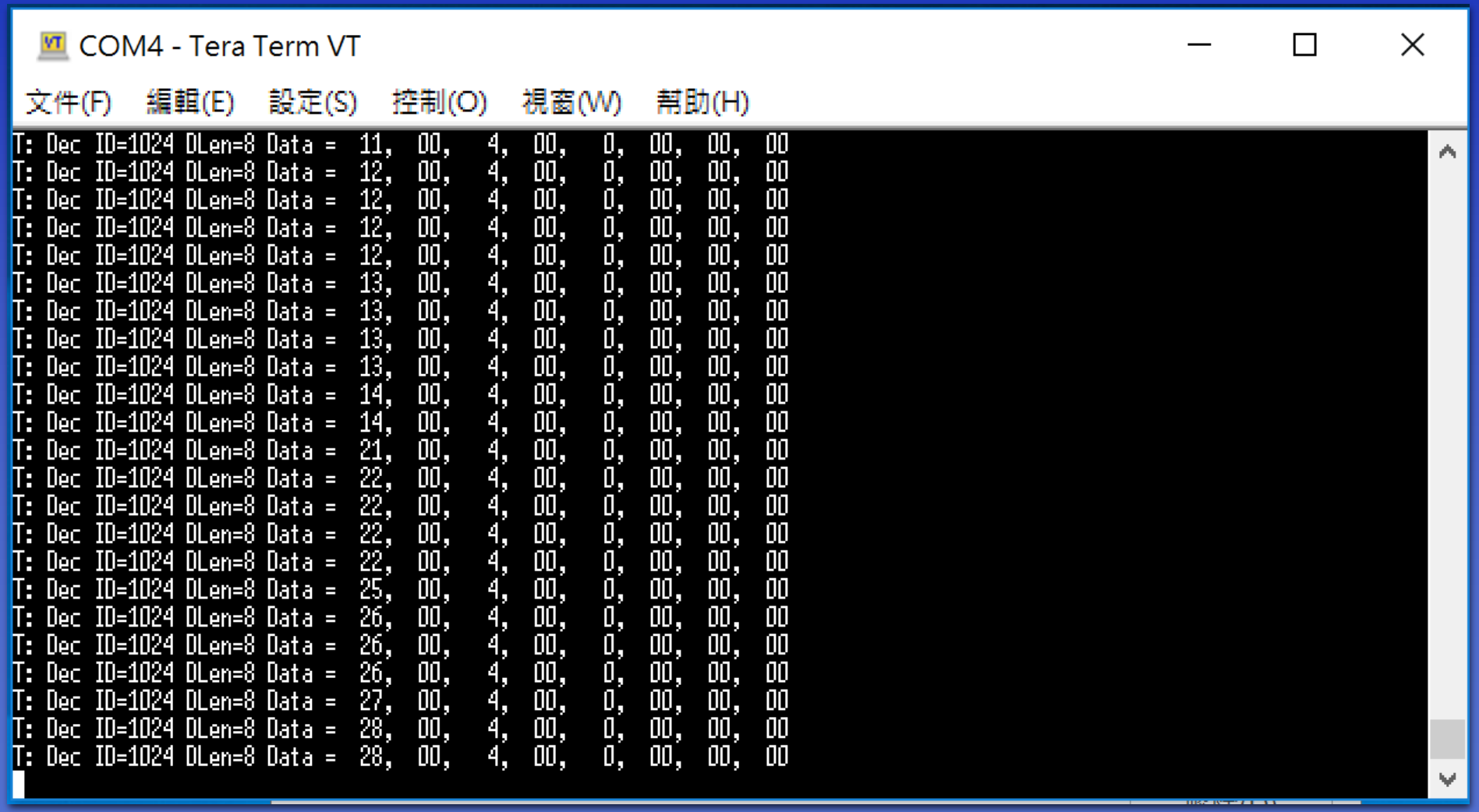
identifier -> SID <10:0> : EID <17:0>

mide = 0 -> Match either standard or extended address message if filters match
mide = 1 -> Match only message types that correpond to 'exide' bit in filter

exide = 0 -> Match messages with standard identifier addresses
exide = 1 -> Match messages with extended identifier addresses

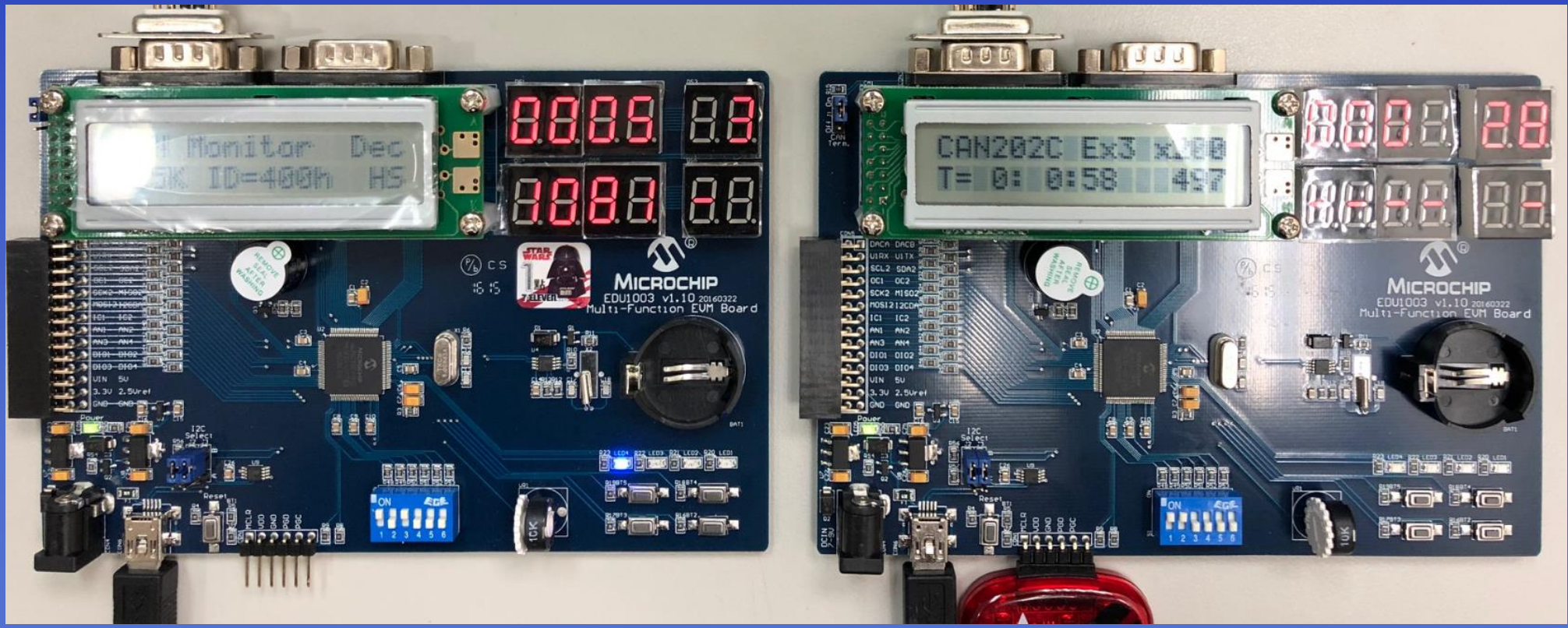
練習三執行結果 (1)

- 按下 EDU1003 BT4 按鍵傳出 Clock 的資訊



練習三執行結果

- 左邊為 Host
  - 按下 EDU1003 BT4 按鍵傳出 Clock 的資訊



Exercise - 4

使用 16 個 Message Buffer 組成的 FIFO 來接收

自主機傳來的 VR1 (類比電壓)信號

練習四 – ECAN FIFO 的使用

- dsPIC33EP ECAN module 可以指定 32 個 buffer 的某些部分要規劃為 FIFO mode
  - 使用 C1FCTRL 設定 ECAN 1 module
  - 必須設定起始點與大小

Register 21-18: CiFCTRL: ECAN FIFO Control Register

R/W-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0	U-0
DMABS<2:0>			—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FSA<4:0>				
bit 7							bit 0

CiFCTRL 暫存器

- FSA 位元決定 FIFO 的起始位址
- DMABS 決定 FIFO 的大小
  - 必須是連續的並且不能與已經使用的 TX or RX Buffer 重疊

bit 15-13	**DMABS<2:0>:** Message Buffer Size bits 111 = Reserved; do not use 110 = 32 buffers in device RAM 101 = 24 buffers in device RAM 100 = 16 buffers in device RAM 011 = 12 buffers in device RAM 010 = Eight buffers in device RAM 001 = Six buffers in device RAM 000 = Four buffers in device RAM
bit 12-5	**Unimplemented:** Read as '0'
bit 4-0	**FSA<4:0>:** FIFO Start Area bits 11111 = Read buffer RB31 11110 = Read buffer RB30 • • • 00010 = TX/RX buffer TRB2 00001 = TX/RX buffer TRB1 00000 = TX/RX buffer TRB0

FIFO 的指位器

- CiFIFO 暫存器存放 FIFO 的指位器
  - FBP : FIFO 現在只向哪一個 Buffer
  - FNRB : FIFO 需要被讀取的 Buffer 號碼
  - 如果 FBP != FNRB 表示 FIFO 中有未被讀取的 Buffer
    - 當讀取完畢並且將該 Buffer 的 RXFUL 清除後 FNRB 位元 會自動被 update

Register 21-19: CiFIFO: ECAN FIFO Status Register

U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
—	—	FBP<5:0>					
bit 15							bit 8

U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
—	—	FNRB<5:0>					
bit 7							bit 0

練習四增加的功能

- 使用 FIFO 來接收 Message ID EXT_VR1_ID 傳送過來的資料 – 資料量比較大 (每 100ms 一次)
  - 使用 Check_CAN_FIFO( ) 副程式檢查並讀出資料
- 將接收的資料讀出後存放於 FRAME_Buffer1 的結構中
- 將 FRAME_Buffer1 中的 DataWord0 的值顯示在七段顯示器的第二行

```
Check_CAN_FIFO();
if ( FRAME_Buffer1.DATA_IN)
{
        EXT_VR1_Value = FRAME_Buffer1.DataWord0 ;
        EXT_ADC_Buf_to_LINE2( ) ;
   FRAME_Buffer1.DATA_IN = 0 ;
}
```

Check_CAN_FIFO( )

- 請注意 RXFUL 位元一定要記得清除

```
void    Check_CAN_FIFO( void )
{
      if ( C1FIFObits.FBP != C1FIFObits.FNRB )
      {
         RXFUL_Var = C1FIFObits.FNRB ;

        FRAME_Buffer1.SID = (ecan1msgBuf[RXFUL_Var][0] >> 2 ) & 0x07ff ;
        if ( FRAME_Buffer1.SID == EXT_VR1_ID  )
        {
           FRAME_Buffer1.DataWord0 = ecan1msgBuf[RXFUL_Var][3] ;          //  Read ADC Value -- It at 1st word
           FRAME_Buffer1.DATA_IN = 1 ;
        }
        RXFUL_Var = 0xfffe ;
        for ( RXFUL_Loop = 0 ; RXFUL_Loop < C1FIFObits.FNRB; RXFUL_Loop++)
           RXFUL_Var <<= 1 ;
           C1RXFUL1 &= RXFUL_Var ;         // Clear RXFUL Flag ...
      }
}
```

練習四對 Filter 設定的注意事項

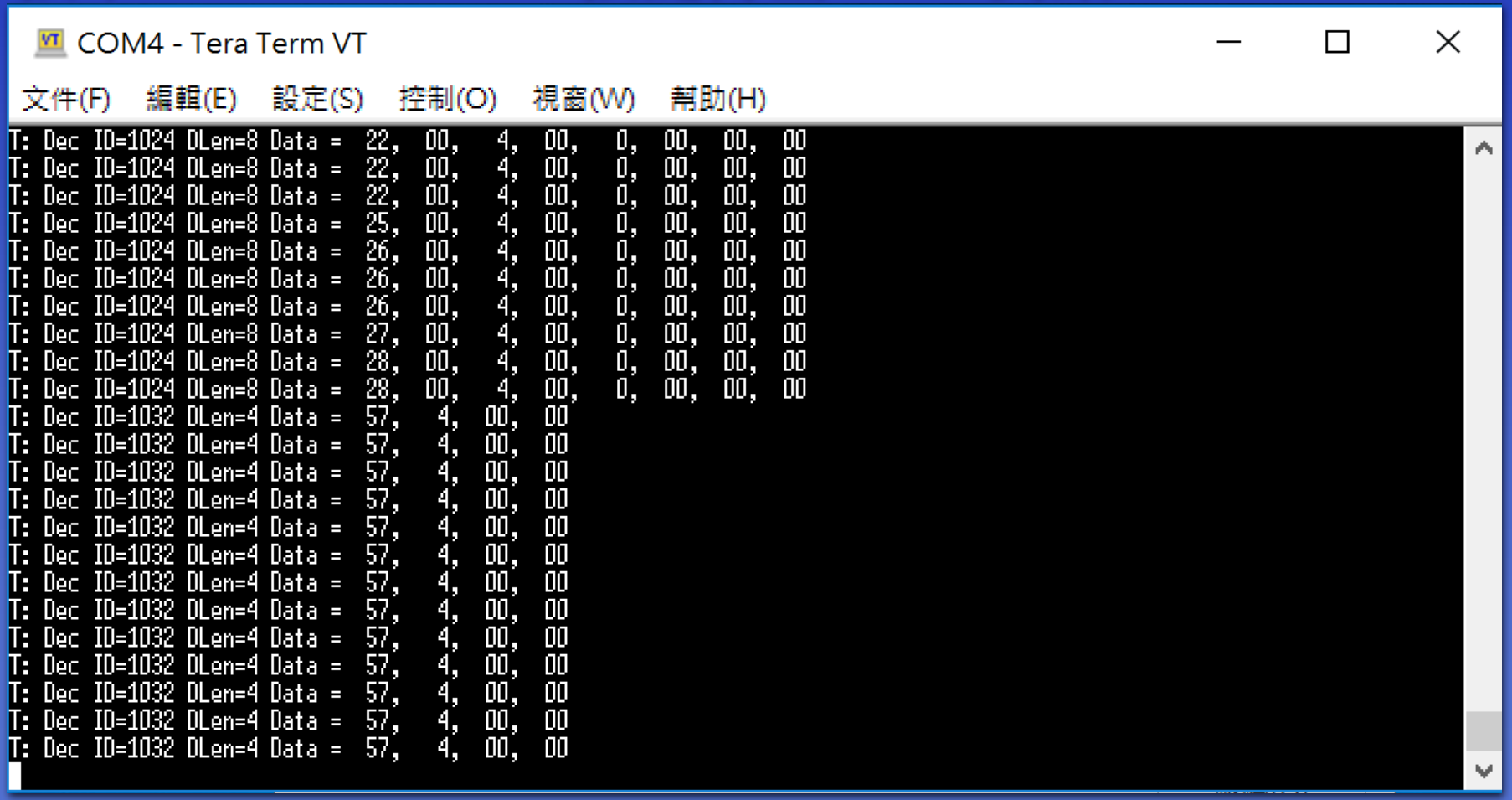
- 再次檢視 Ecan1_config.c 中檢查 Ecan1Init( ) 副程式中是否有對 Filter 以及 Mask 做正確設定
  - Filter 2 被設定過濾 EXT_VR1_ID
  - Filter 2 被指定將過濾後的資料寫入 FIFO
  - Filter 2 被指定使用 Mask 1 來決定哪些位元要被檢查
  - Mask1 被設定為檢查每一個位元
  - EXT_VR1_ID 在 common.h 中被宣告

```
Ecan1WriteRxAcptFilter( 1, (0x00 | EXT_CLOCK_ID), 0, 04, 1 );
Ecan1WriteRxAcptFilter( 2, (0x00 | EXT_VR1_ID), 0, 15, 1 );

Ecan1WriteRxAcptMask( 1, 0x1FFFFFFF, 1, 0 );
```

練習四執行結果 (1)

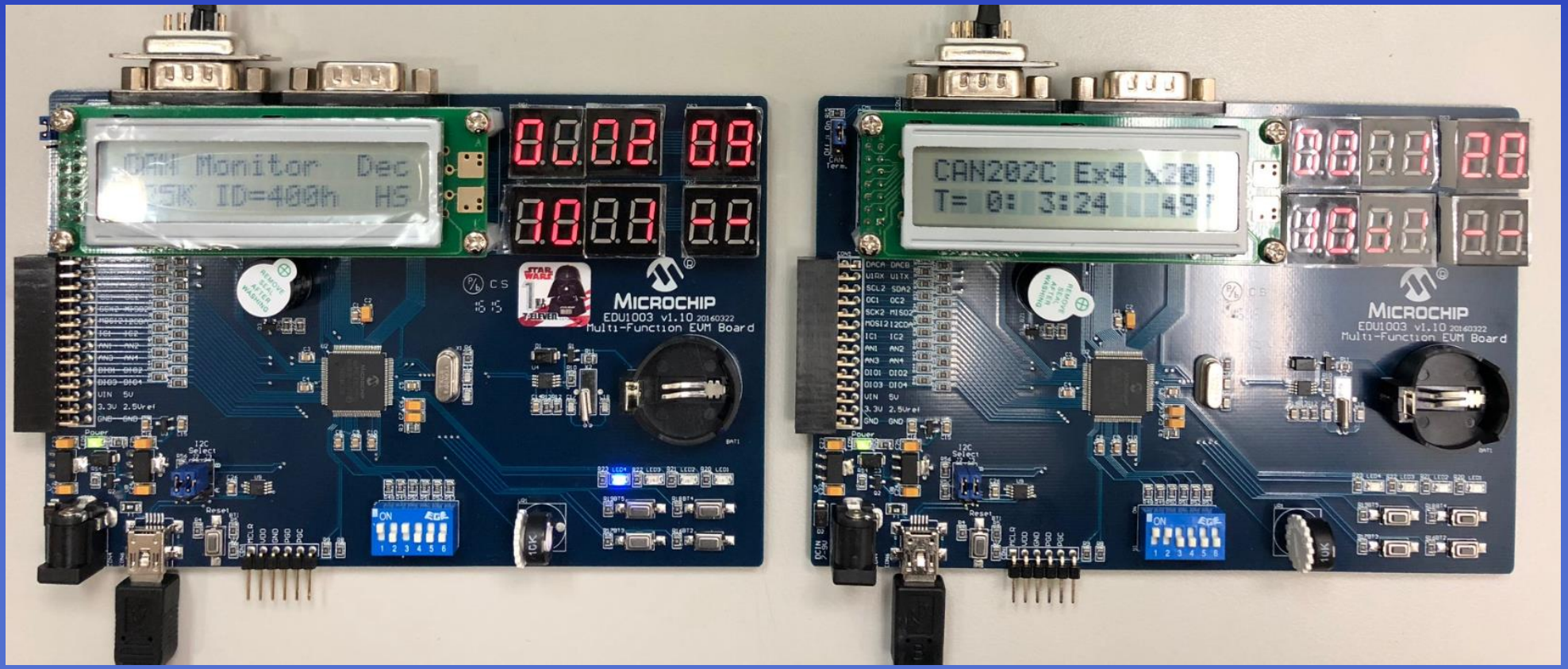
- 按下 EDU1003 BT4 按鍵傳出 Clock 的資訊
- 按下 EDU1003 BT2 按鍵傳出 VR1 資訊



```
T: Dec ID=1024 DLen=8 Data =  22,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  22,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  22,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  25,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  26,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  26,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  26,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  27,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  28,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1024 DLen=8 Data =  28,  00,   4,  00,   0,  00,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
T: Dec ID=1032 DLen=4 Data =  57,   4,  00,  00
```

練習四執行結果 (2)

- 左邊為 Host
  - 按下 EDU1003 BT4 按鍵傳出 Clock 的資訊
  - 按下 EDU1003 BT2 按鍵傳出 VR1 資訊



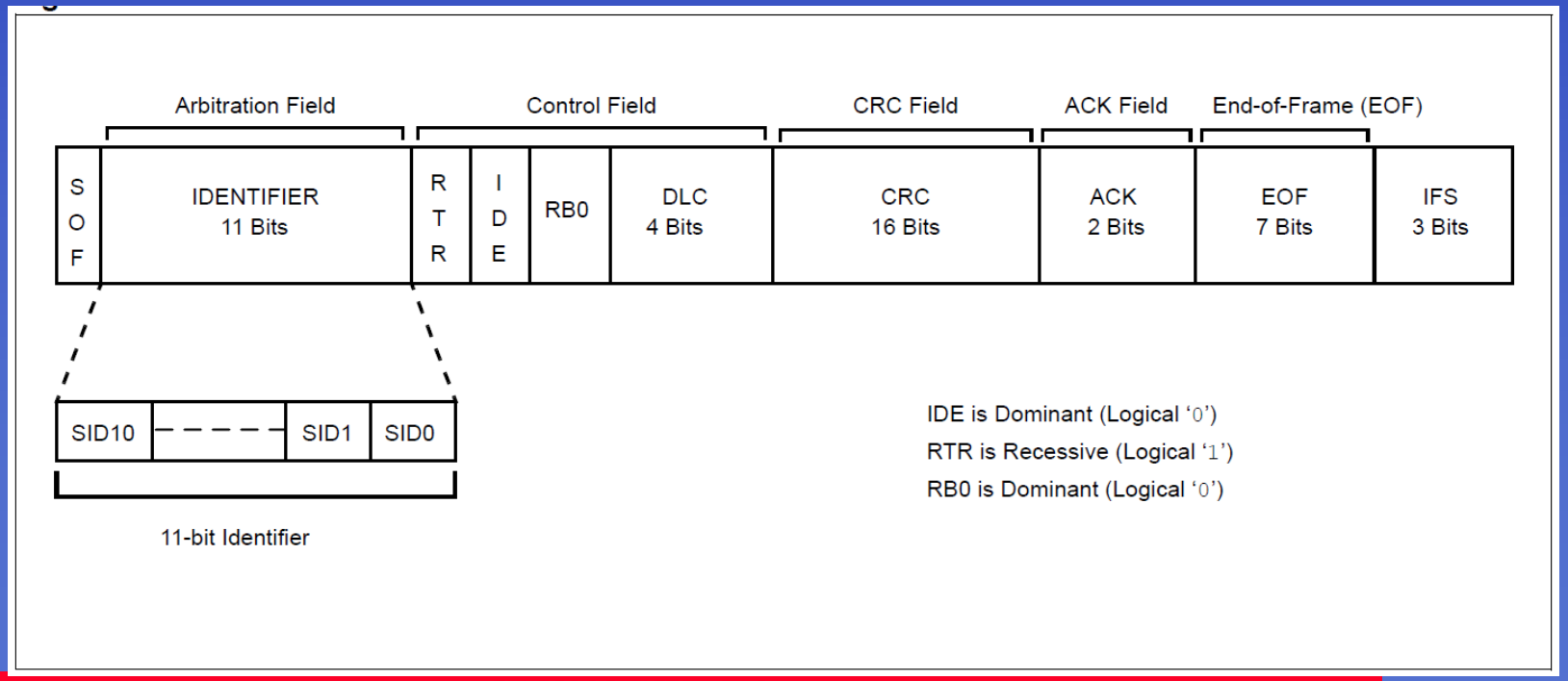
Exercise - 5

dsPIC33EP CAN Module

自動的 Remote Request 回應

RTR Frame 的格式

- 複習 Remote Transfer Request 的不同地方
  - 以 Standard Frame 為例
  - RTR 位元需設為 Recessive
  - 資料長度 DLC 設定為 “0” – 因為是要向對方索取資料



回應 Remote Transfer Request

- 一般的 CAN Controller 需要做正確的 Filter 設定，在收到 RTR Frame 後，以程式來驅動資料的回送
- dsPIC33EP 的 CAN Engine 有自動回應 RTR 的功能，但要遵循使用手冊的指引來正確設定
  - ECAN Module Reference Manual 的 21.6.3.2
  - 重點：專用的 Acceptance Filter 及 ID 設定、指向正確的 Message Buffer、設定 TXEN 以及 RTREN、要搭配獨立的 Mask !

21.6.3.2 RESPOND TO A REMOTE FRAME

The node acting as the source to respond to the remote frame request needs to configure an acceptance filter to match the identifier of the Remote Frame. Message buffers 0-7 can respond to remote frames, so the Acceptance Filter Buffer Pointer (FnBP) should point to one of the eight message buffers. The TX/RX Buffer Selection (TXENn) and Auto-Remote Transmit Enable (RTRENm) bits in the ECAN Transmit/Receive Control register (CiTRMNCON<7> and CiTRmnCON<2>) must be set to respond to the Remote Frame.

This is the only case where the Acceptance Filter Buffer Pointer (FnBP) points to a message buffer that is configured for transmission (TXENn = 1).

Auto RTR Response 的程式範例 (1)

- ecan1_config.c 對 Filter & Mask 的設定
  - 以 Filter 0 來接收對 0x200 的 RTR，使用 Buffer 2

```
Ecan1WriteRxAcptFilter( 1, (0x00 | EXT_CLOCK_ID), 0, 04, 1 );          // Filter 1
Ecan1WriteRxAcptFilter( 2, (0x00 | EXT_VR1_ID ) , 0, 15, 1 );          // Filter 2
Ecan1WriteRxAcptFilter( 0, 0x200 , 0, 02, 0 );                          // Filter 0 point to B
/*    Mask Configuration
Ecan1WriteRxAcptMask(int m, long identifierMask, unsigned int mide, unsigned int exide)
m = 0 to 2 -> Mask Number
identifier -> SID <10:0> : EID <17:0>
mide = 0 -> Match either standard or extended address message if filters match
mide = 1 -> Match only message types that correpond to 'exide' bit in filter

exide = 0 -> Match messages with standard identifier addresses
exide = 1 -> Match messages with extended identifier addresses
*/
Ecan1WriteRxAcptMask( 1, 0x1FFFFFFF, 1, 0 );
Ecan1WriteRxAcptMask( 0, 0x1FFFFFFF, 1, 0 );
```

Auto RTR Response 的程式範例 (2)

- ecan1_config.c 對 Message Buffer 的設定
  - Message Buffer2 的 TXEN & RTREN 都要被設定

```
/* ECAN transmit/receive message control */
C1RXFUL1 = C1RXFUL2 = C1RXOVF1 = C1RXOVF2 = 0x0000;
C1TR01CONbits.TXEN0 = 1;      /* ECAN1, Buffer 0 is a Transmit Buffer */
C1TR01CONbits.TXEN1 = 1;      /* ECAN1, Buffer 1 is a Transmit Buffer */
C1TR23CONbits.TXEN2 = 1;
C1TR23CONbits.RTREN2 = 1 ;    // Set TX Buffer0 as Auto response to Remote Transfer

C1TR01CONbits.TX0PRI = 0b11;  /* Message Buffer 0 Priority Level */
C1TR01CONbits.TX1PRI = 0b11;  /* Message Buffer 0 Priority Level */
C1TR23CONbits.TX2PRI = 0b11;  /* Message Buffer 0 Priority Level */
```

Auto RTR Response 的程式範例 (3)

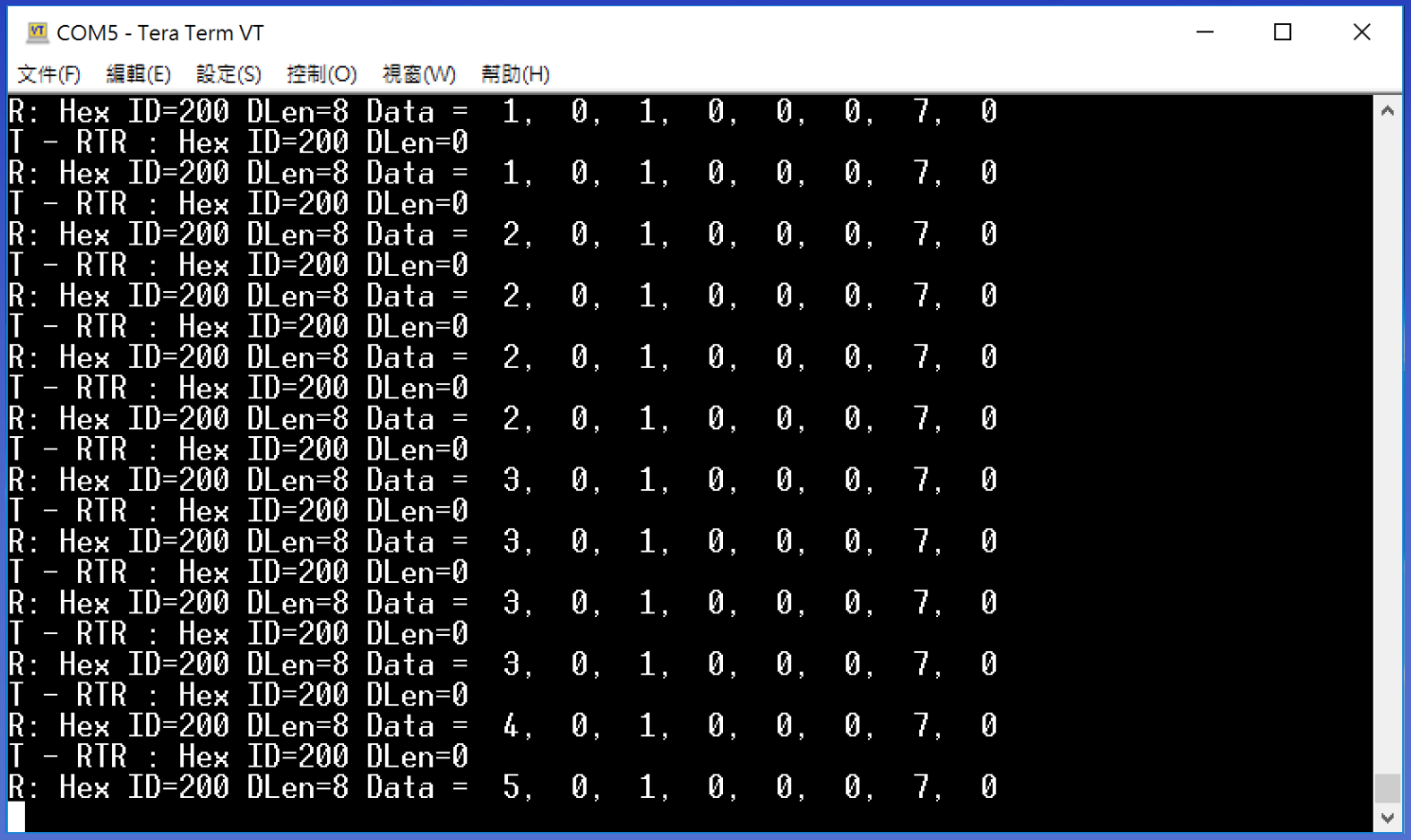
- 每一秒鐘的時間到達，主程式執行 CAN_Send-CLOCK_Remote( )
- 請注意，我們只檢查 TXREQ2，但填完資料部會設定它
  - dsPIC33EP 的 CAN Engine 會在收到對 CLOCK_ID (0x200) 的 RTR Frame 時自動設定 TXREQ2

```
void    CAN_Send_CLOCK_Remote(void)
{
   if ( ! C1TR23CONbits.TXREQ2 )
   {
          Ecan1WriteTxMsgBufId( 2, CLOCK_ID, 0, 0 );
          Ecan1WriteTxMsgBufData( 2, 8,  (unsigned int) CLOCK1.Second,
                                         (unsigned int) CLOCK1.Minute,
                                         (unsigned int) CLOCK1.Hour ,
                                         (unsigned int) BOARD_ID.ID_Byte);
   }

}
```

練習五的執行結果

- 按下 CAN Bus Debugger 的 BT3, 結果如下
  - CAN Node 會將 Clock 以 0x200 的 ID 送出



Any Question ?

謝謝 參加 CAN202C !