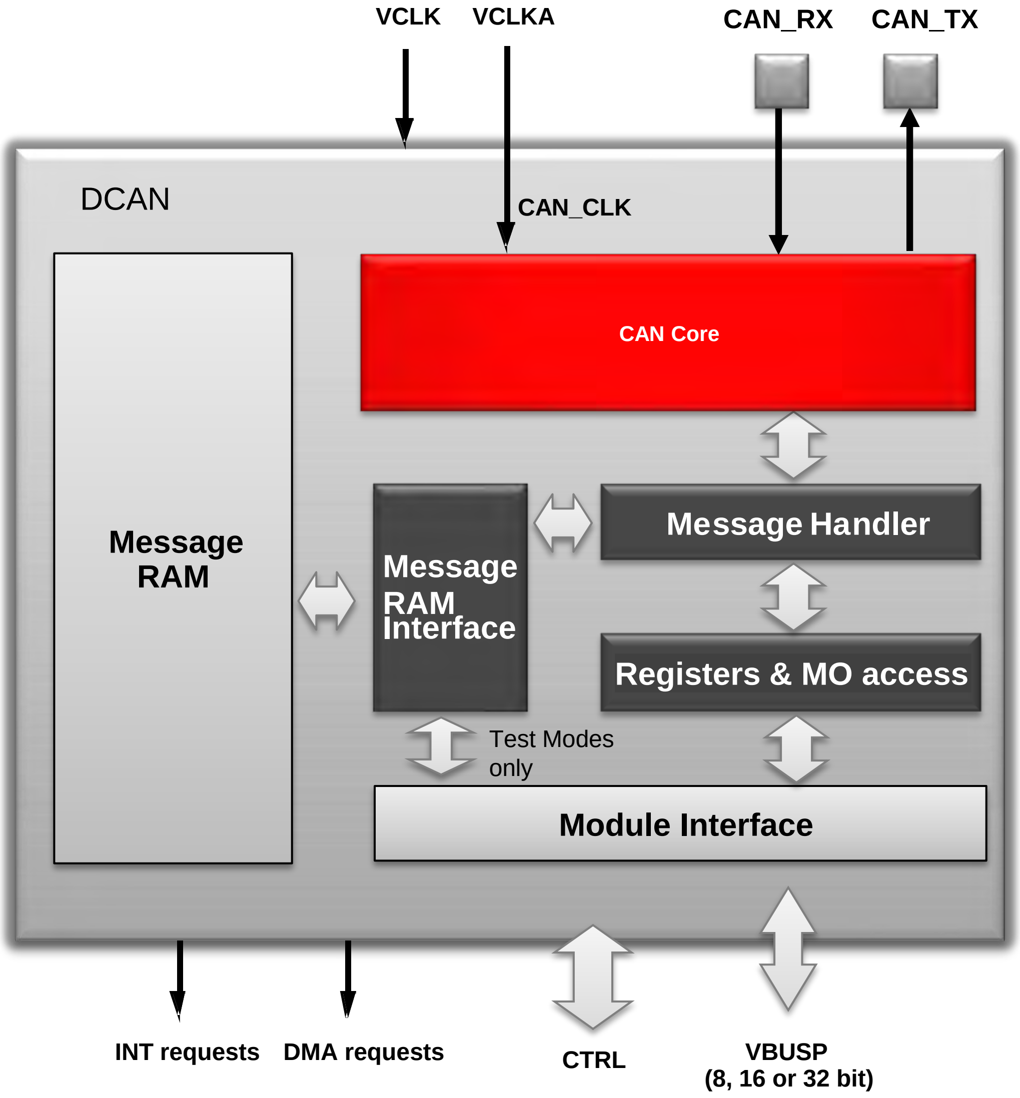
TMS470M: Controller Area Network (DCAN)

DCAN Architecture



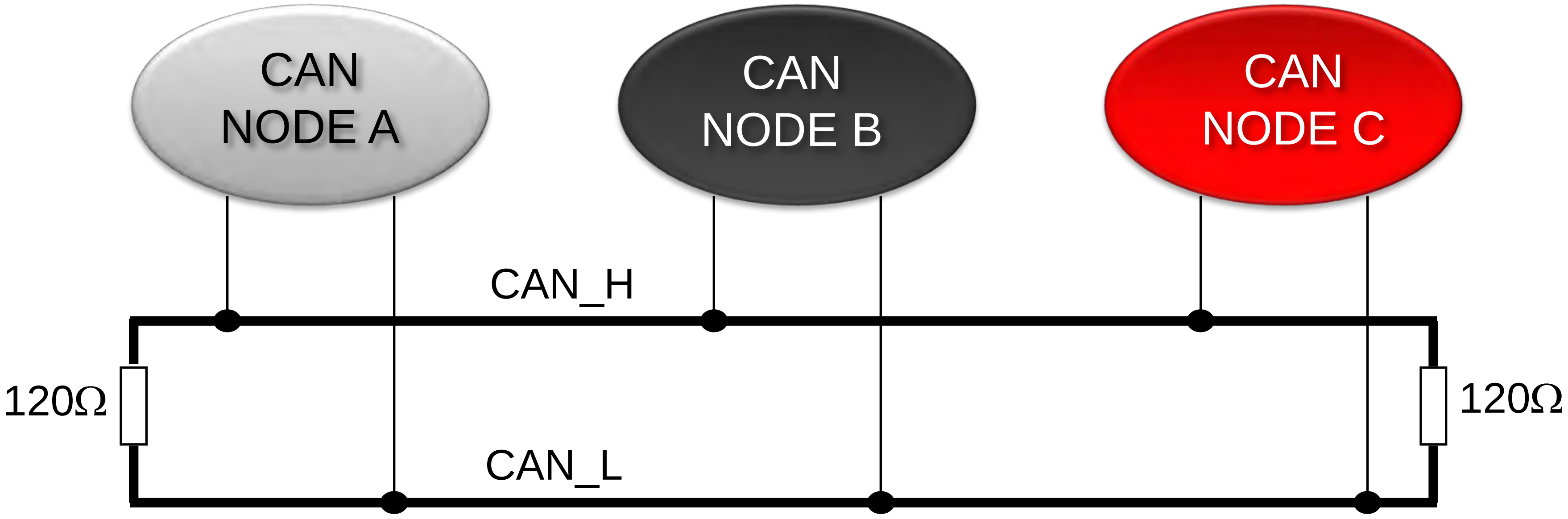
- Full CAN according to protocol version 2.0 part A, B
- CAN Core
  - handles all CAN protocol functions
- Message Handler
  - controls data transfer between CAN Core, Message Interface registers and RAM
  - handles acceptance filtering and interrupts
- Message RAM
  - 16 mailboxes (DCAN 1)
  - 16 mailboxes (DCAN 2)
- Registers and Message Object access (IFx)
  - Status and configuration registers for module setup and indirect Message Object access through interface registers (IFx)
- Module Interface
  - 32-bit interface to VBUS clock domain

DCAN Features Overview

- Full CAN according to protocol version 2.0 part A, B
- Standard and Extended Identifiers
- Programmable Bit Timing, Bit rates up to 1 MBit/s
- Up to 16 Message Objects (MO) / Mailboxes
- Identifier Masks for each Message Object
- Programmable FIFO mode for Message Objects
- Dual clock feature
- Possible automatic retransmission of a frame in case of lost arbitration or error
- Bus diagnostic: Bus off, Bus error passive, Bus error warning, Bus stuck dominant
- Frame error report: CRC, Stuff, Form, Bit and Acknowledgement errors
- Programmable loop-back modes for self-test operation
- Suspend modes for debug support
- Parity check mechanism for all RAM modules

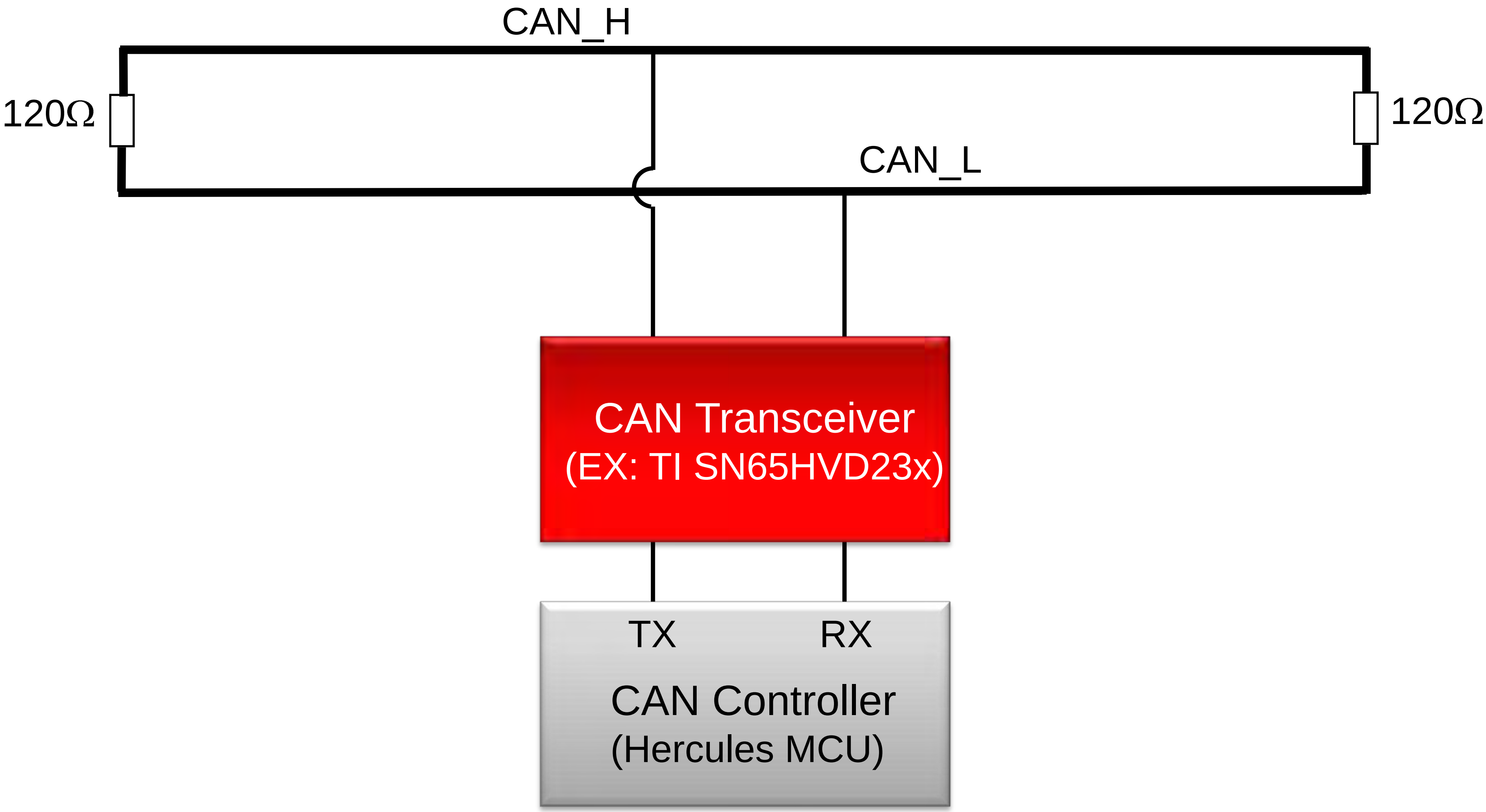
CAN Bus

- Two wire differential bus (usually twisted pair)
- Max. bus length depend on transmission rate
  - 40 meters @ 1 Mbps



CAN Node

Wired-AND Bus Connection



CAN_H

120Ω

120Ω

CAN_L

CAN Transceiver
(EX: TI SN65HVD23x)

TX

RX

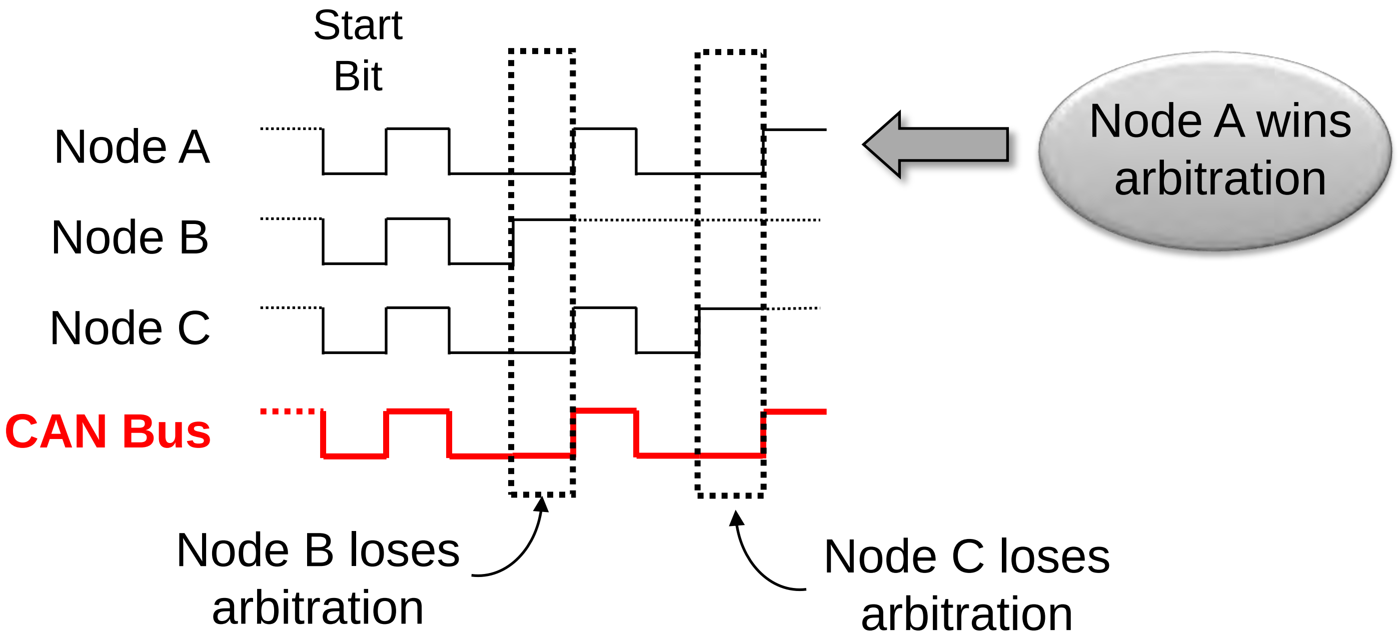
CAN Controller
(Hercules MCU)

Principles of CAN Network Operation

- Data messages transmitted are identifier based, not address based
- Content of message is labeled by an identifier that is unique throughout the network
  - (e.g. rpm, temperature, position, pressure, etc.)
- All nodes on network receive the message and each performs an acceptance test on the identifier
- If message is relevant, it is processed (received); otherwise it is ignored
- Unique identifier also determines the priority of the message
  - (lower the numerical value of the identifier, the higher the priority)
- When two or more nodes attempt to transmit at the same time, a non-destructive arbitration technique guarantees messages are sent in order of priority and no messages are lost

Non-Destructive Bitwise Arbitration

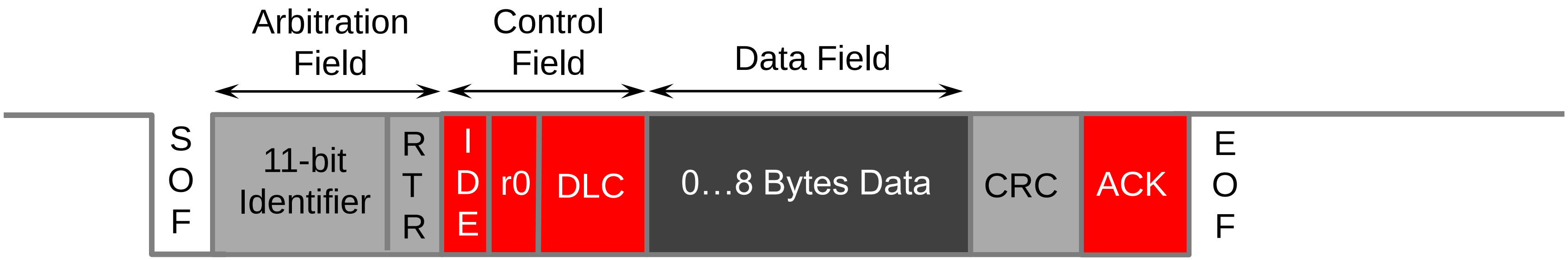
- Bus arbitration resolved via arbitration with wired-AND bus connections
  - Dominate state (logic 0, bus is high)
  - Recessive state (logic 1, bus is low)



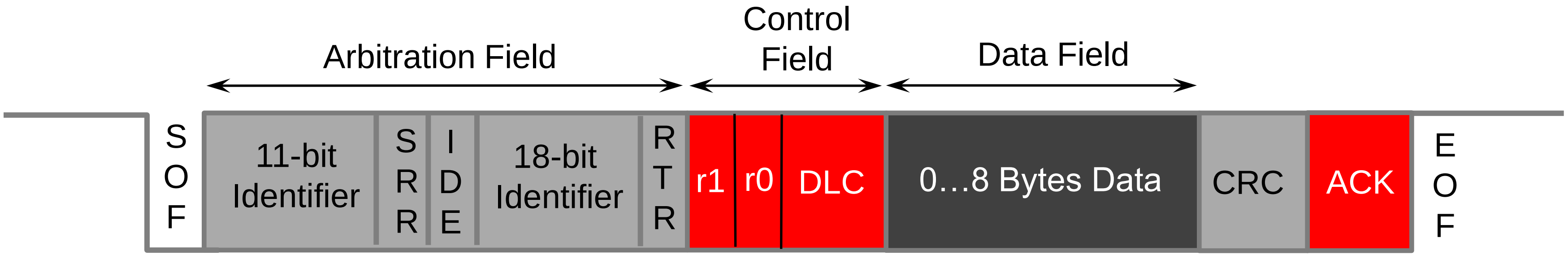
CAN Message Format

- Data is transmitted and received using Message Frames
- 8 byte data payload per message
- Standard and Extended identifier formats

Standard Frame: 11-bit Identifier (CAN v2.0A)

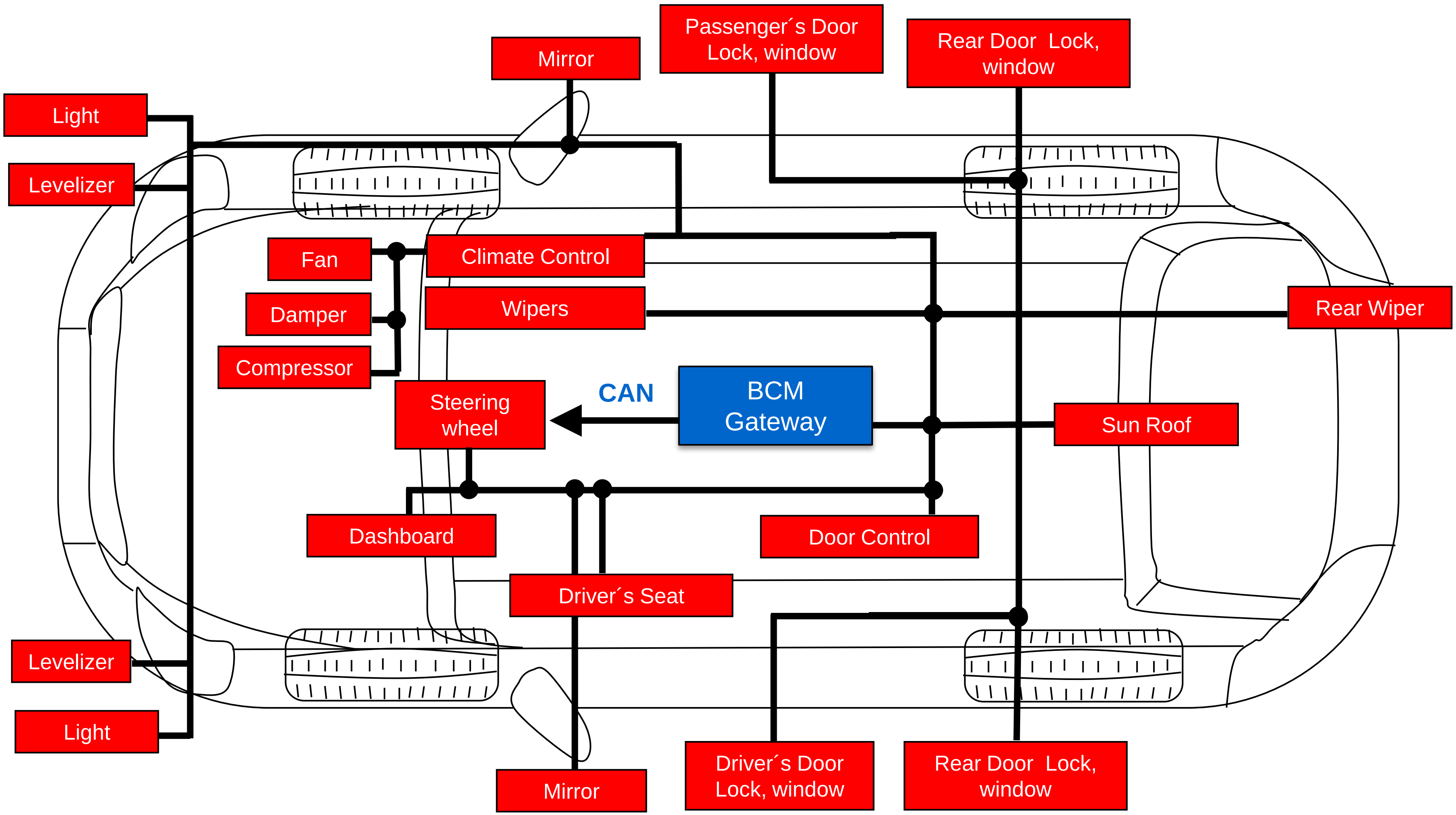


Extended Frame: 29-bit Identifier (CAN v2.0B)



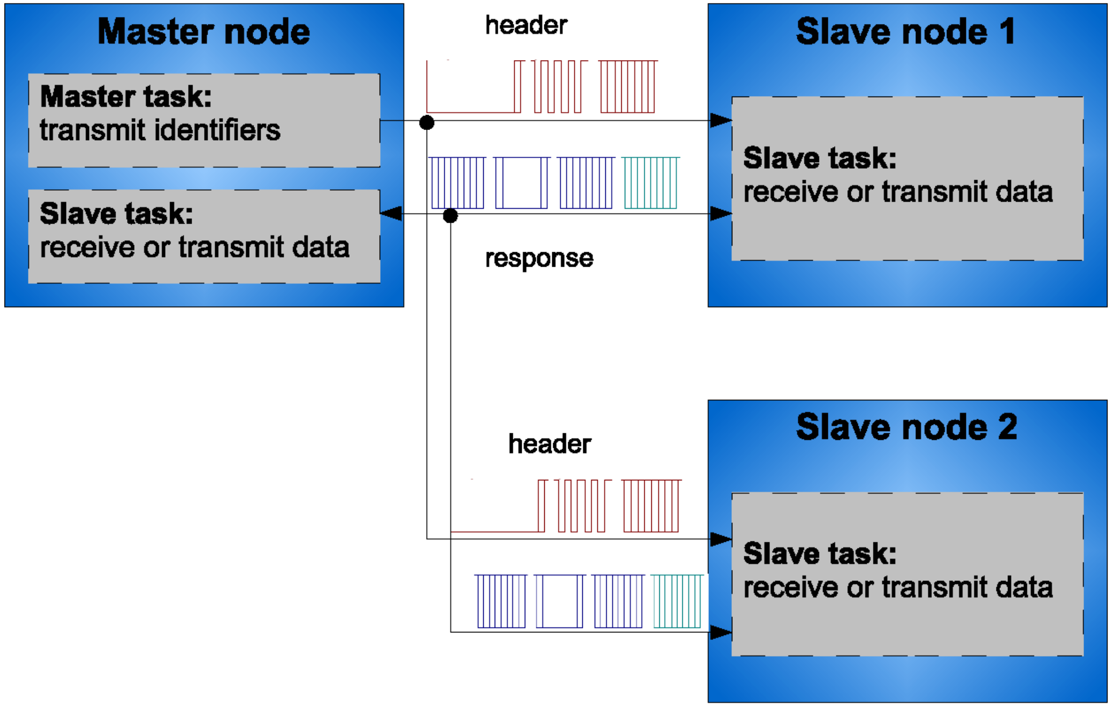
TMS470M: Local Interconnect Network (LIN)

Typical LIN Applications

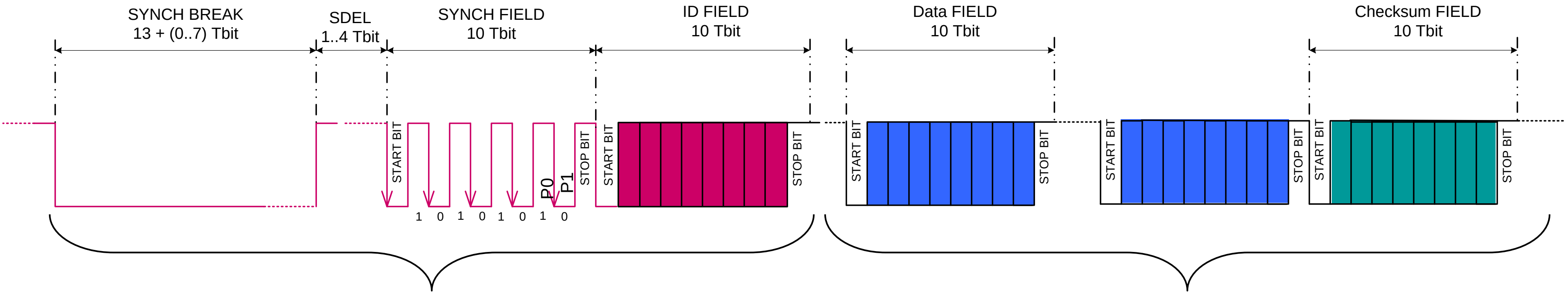


LIN Communication Concept

- Single Master concept with max. 16 nodes in one LIN cluster
- LIN supports baud rates from 1 to 20KHz
- Single wire low cost bus system often used as a sub network to comfort CAN.



LIN Message Frame



MASTER

- Synch break signalling beginning of a new message
- Synch field: 0x55
- ID Field:

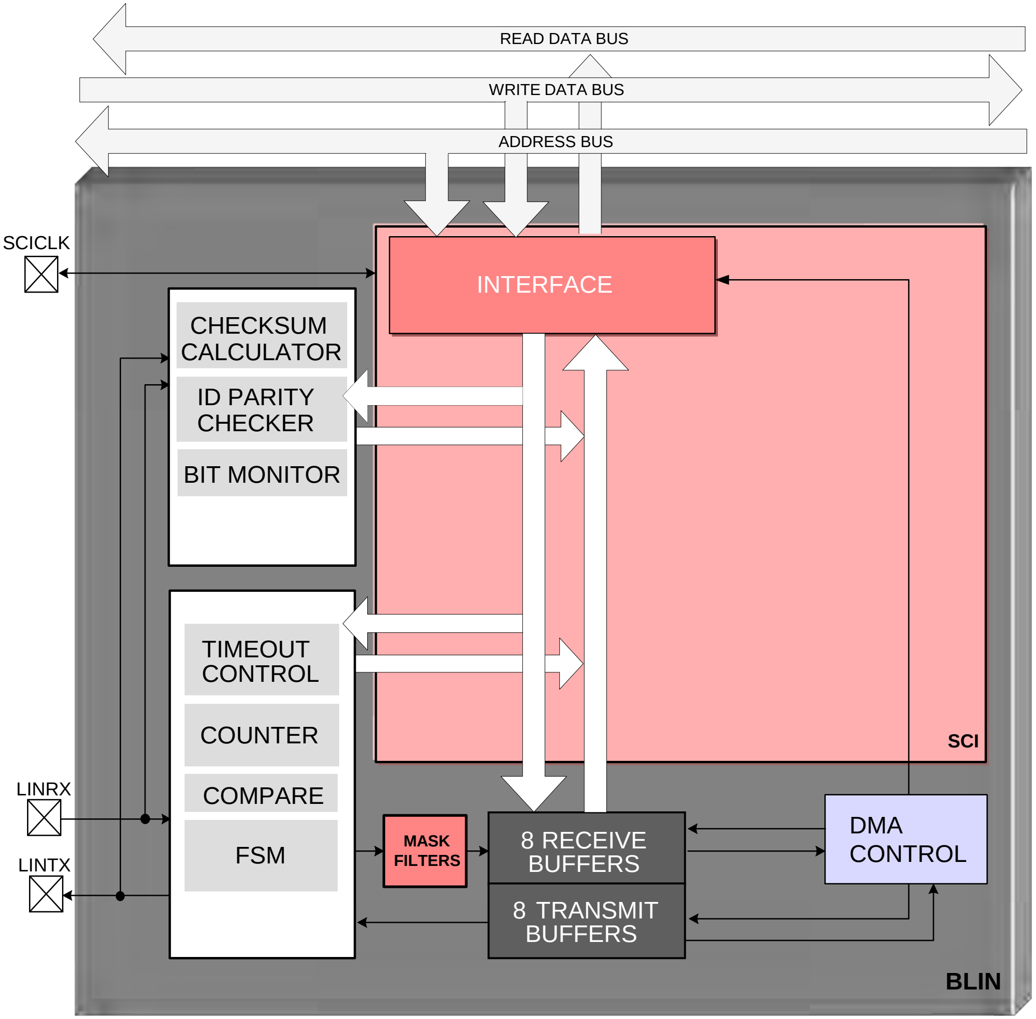
  ID 0x00 to 0x3F (0 to 63)
- Diagnostics:

  ID 0x3C (master request)
  ID 0x3D (slave response)
- User Defined: ID 0x3E

SLAVE

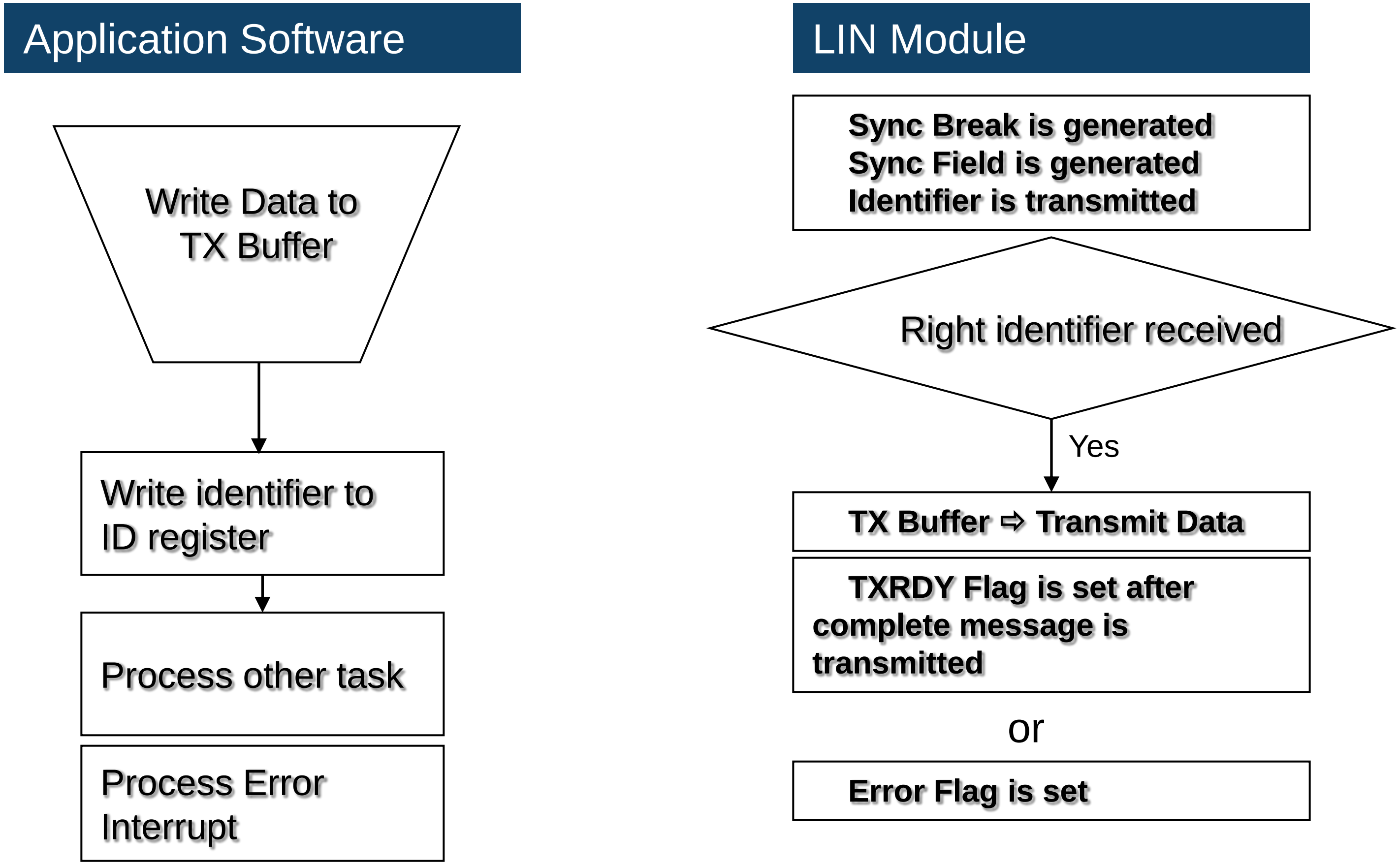
- Response with 0 to 8 data fields
- Checksum field
  - classic CS LIN1.3
    - over data bits only
  - enhanced CS LIN2.0
    - over data bits and the protected identifier

LIN Key Features



- Compatible with LIN 1.3 or 2.0
- LIN 2.0 Master Compliant
- HW LIN protocol handler
  - Multi-buffered receive and transmit units
  - Automatic checksum generation and validation
  - ID masks for message filtering
  - DMA capability
- Synch break detection
- Slave automatic synchronization
- Optional baud rate update
- Synchronization validation
- Automatic bit monitoring
- Automatic error detection
- SCI (UART) mode
  - Max 3.125Mbps with 100MHz VCLK

LIN: Master Transmission



Application Software

Write Data to TX Buffer

Write identifier to ID register

Process other task

Process Error Interrupt

LIN Module

Sync Break is generated
Sync Field is generated
Identifier is transmitted

Right identifier received

Yes

TX Buffer ⇨ Transmit Data

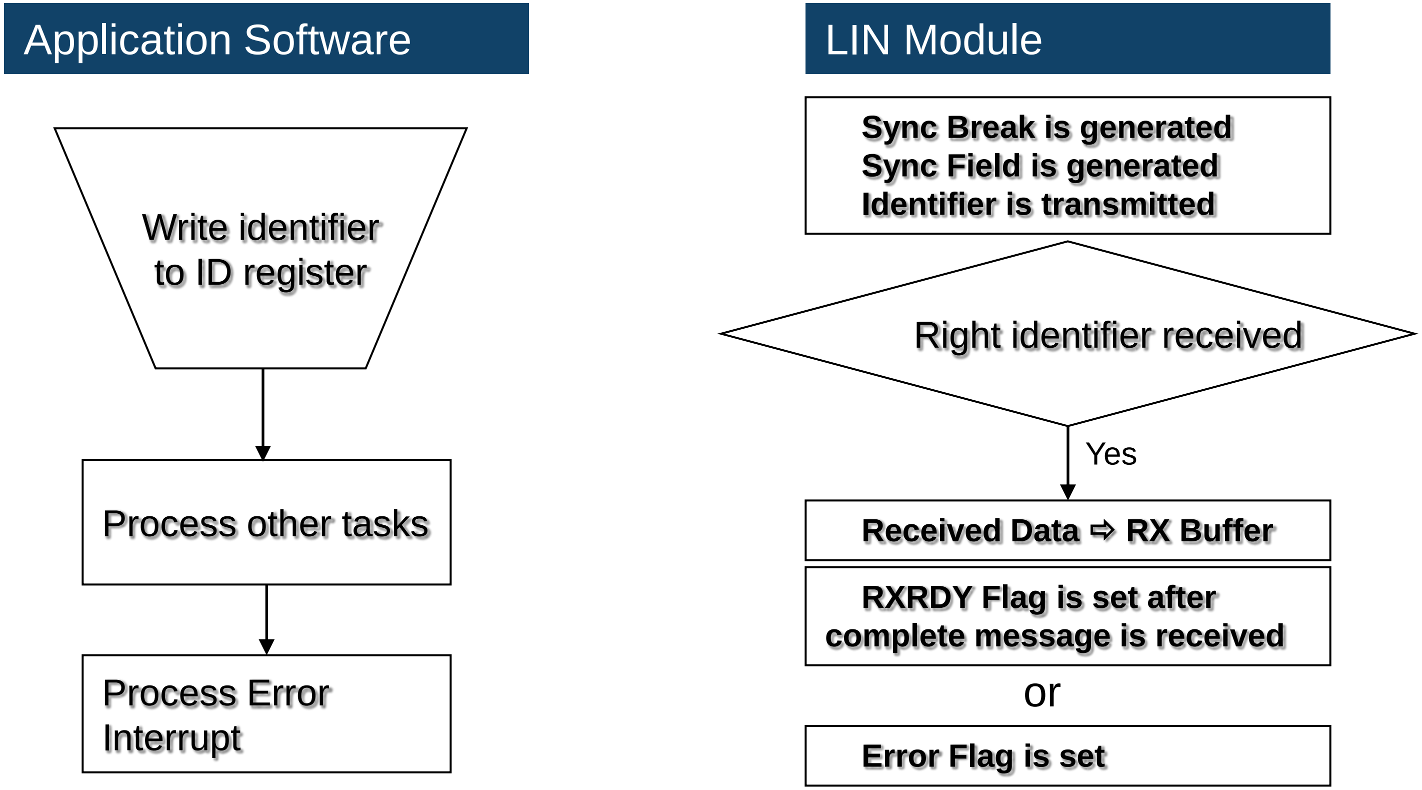
TXRDY Flag is set after complete message is transmitted

or

Error Flag is set

- Application software handles the preparation of transmitted data and starts the transmission with writing the ID.

LIN: Master Reception


Application Software
Write identifier to ID register
Process other tasks
Process Error Interrupt
LIN Module
Sync Break is generated
Sync Field is generated
Identifier is transmitted
Right identifier received
Yes
Received Data ⇨ RX Buffer
RXRDY Flag is set after complete message is received
or
Error Flag is set

- Application software starts the transmission with writing the ID and handles the received data.

LIN – SCI Mode Features

- Programmable Frame Format

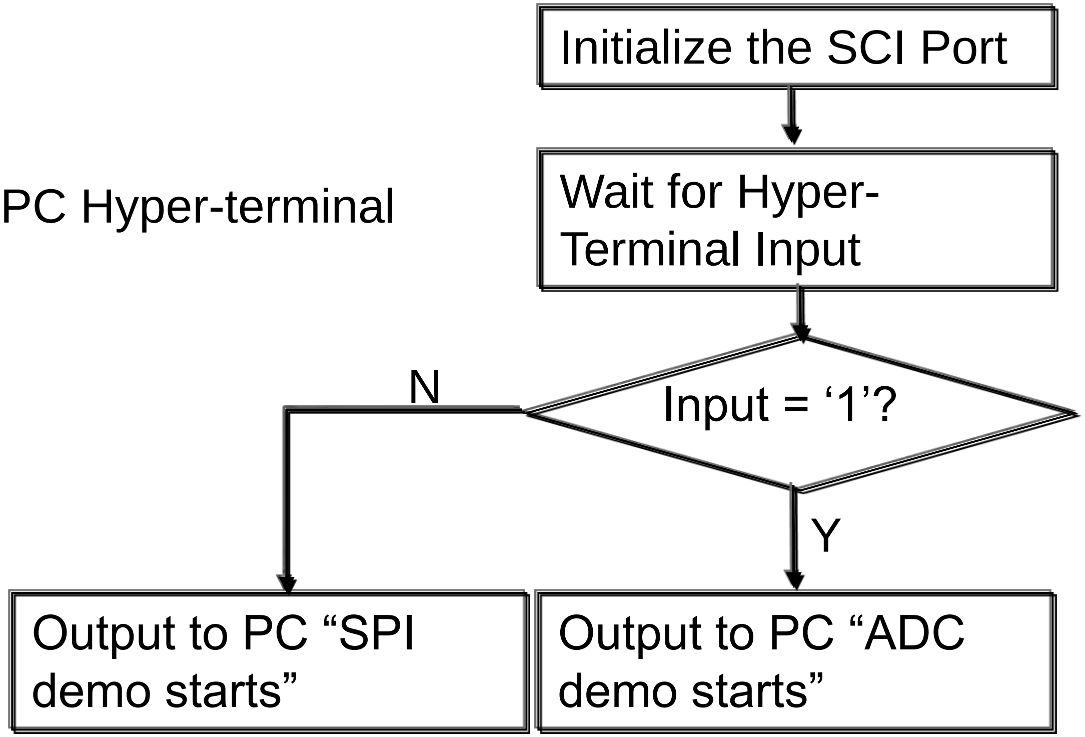
1	Start Bit
1 to 8	Data Bits
0 or 1	Address Bit
0 or 1	Parity Bit
1 or 2	Stop Bits

- Asynchronous Communications Format
- 2 Multiprocessor Modes with Wake-up Capability
  Idle-Line Mode; Address-Bit Mode
- Programmable Baud Rate
  - more than 16 700 000 different Baud Rates
- Error Detection
  - Parity, Overrun and Framing Error
  - Break Detect
- Noise Protection Capability
- Double-buffered Receive and Transmit Function

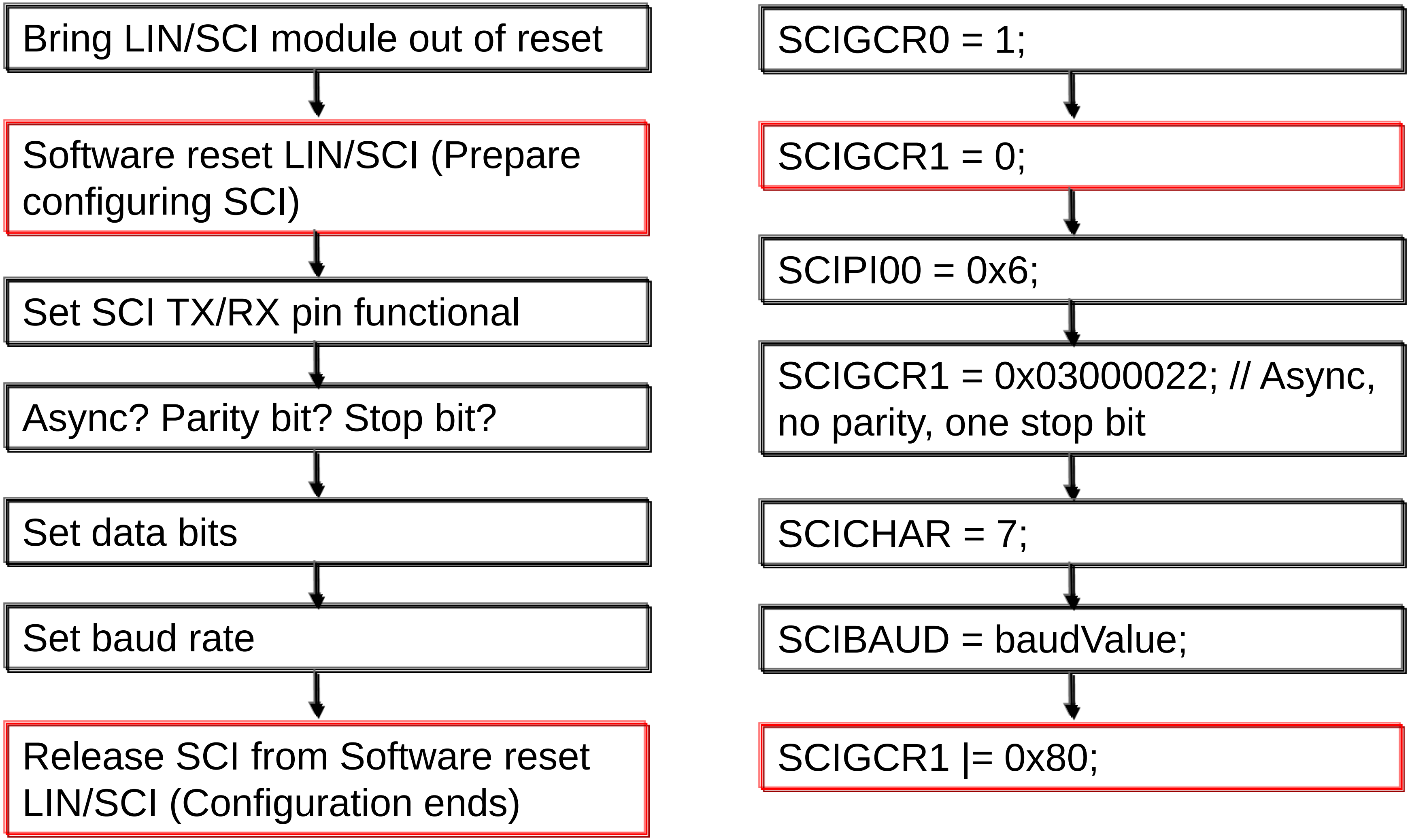
LAB3: PC Communication Using SCI

Overview

- In this project we will:
  - Setup the SCI module
  - Input from and output to PC Hyper-terminal


Initialize the SCI Port
Wait for Hyper-Terminal Input
Input = ‘1’?
N
Y
Output to PC “SPI demo starts”
Output to PC “ADC demo starts”

Initializing the SCI Module



Bring LIN/SCI module out of reset

Software reset LIN/SCI (Prepare configuring SCI)

Set SCI TX/RX pin functional

Async? Parity bit? Stop bit?

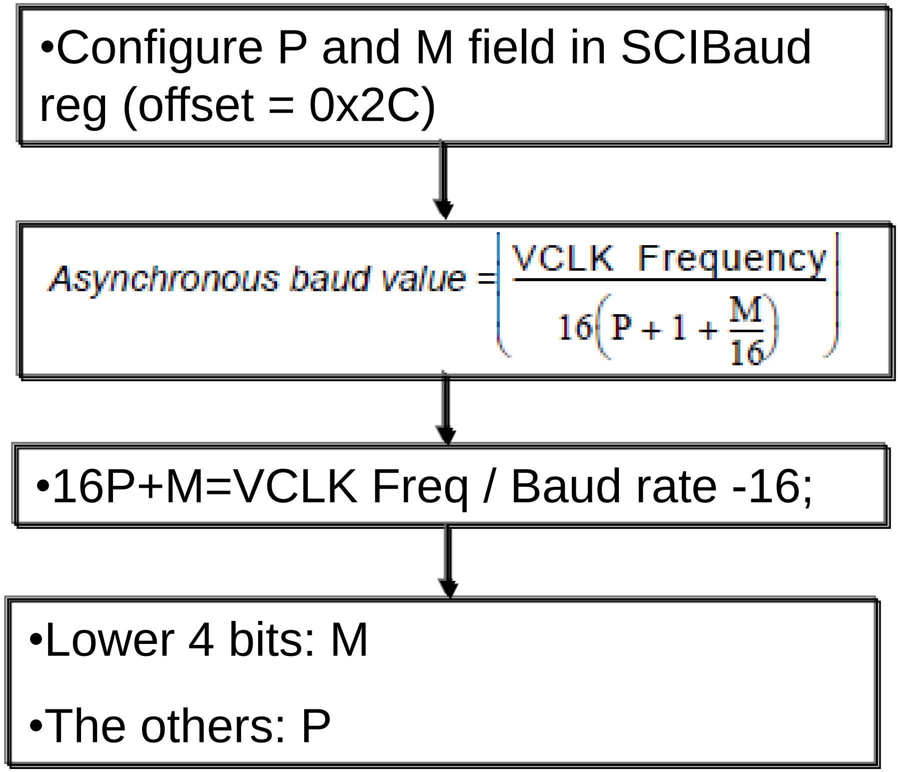
Set data bits

Set baud rate

Release SCI from Software reset LIN/SCI (Configuration ends)

```
SCIGCR0 = 1;
SCIGCR1 = 0;
SCIPI00 = 0x6;
SCIGCR1 = 0x03000022; // Async, no parity, one stop bit
SCICHAR = 7;
SCIBAUD = baudValue;
SCIGCR1 |= 0x80;
```

Setting baud rate



- Configure P and M field in SCIBaud reg (offset = 0x2C)

$$\textit{Asynchronous baud value} = \frac{\text{VCLK Frequency}}{16\left(P + 1 + \frac{M}{16}\right)}$$

- 16P+M=VCLK Freq / Baud rate -16;

- Lower 4 bits: M

- The others: P

- 80MHz VCLK

temp = 80000 / 19.2-16;

((temp & 0xF)<<24)
(temp>>4);

Input from / Output to PC hyper-terminal

- Input: Read data from register SCIRD
- Output: Write data to register SCITD
- Before read/write, user shall check SCIFLR register to see whether TX buffer is ready to be written or Rx buffer contains new data to be read.

Communicating with the SCI Module

- In Code Composer Studio, insert the following into User Code 1 in main.c

```
/* USER CODE BEGIN (1) */
            //Configure the Baud rate to 19.2K.
            temp = 80000 / 19.2-16;
            temp = ((temp & 0xF)<<24) | (temp>>4);
            SCI_Init(temp); // Initialize to 19200bps, 8N1
            PutText("SCI UART IS SET.\r");
            PutText("Press 1 for ADC DEMO; Press 2 for SPI DEMO\r");

            while((ADC_SPI=GetChar())==0);
            if(ADC_SPI=='1') //ADC DEMO
            {           PutText("ADC demo starts.\r");
                        /* USER CODE start (ADC) */
                        /* USER CODE end (ADC) */
            }
            else //SPI DEMO
            {           PutText("SPI demo starts.\r");
                        /* USER CODE start (SPI) */
                        /* USER CODE end (SPI) */
            }
/* USER CODE END (1) */
```

Initializing the Modules

- Insert the following into User Code 2 in SCI.C

```
/* USER CODE BEGIN (2) */
SCIGCR0 = 1;        // Module Out OF Reset
SCIGCR1 = 0;         //SWnRST = 0, Clears all Flag and can config
SCIPI00 = 0x6;       // TX/RX Pin Functional
SCIGCR1 = 0x03000022; // Async, no parity, one stop bit
SCICHAR = 7;          // 8 data bits,
SCIBAUD = baudValue; // Configure baud rate
SCIGCR1 |= 0x80;
// SWnRST = 1, SCI Config is done and should not be disturbed
/* USER CODE END (2) */
```

Building and Deploying the Code

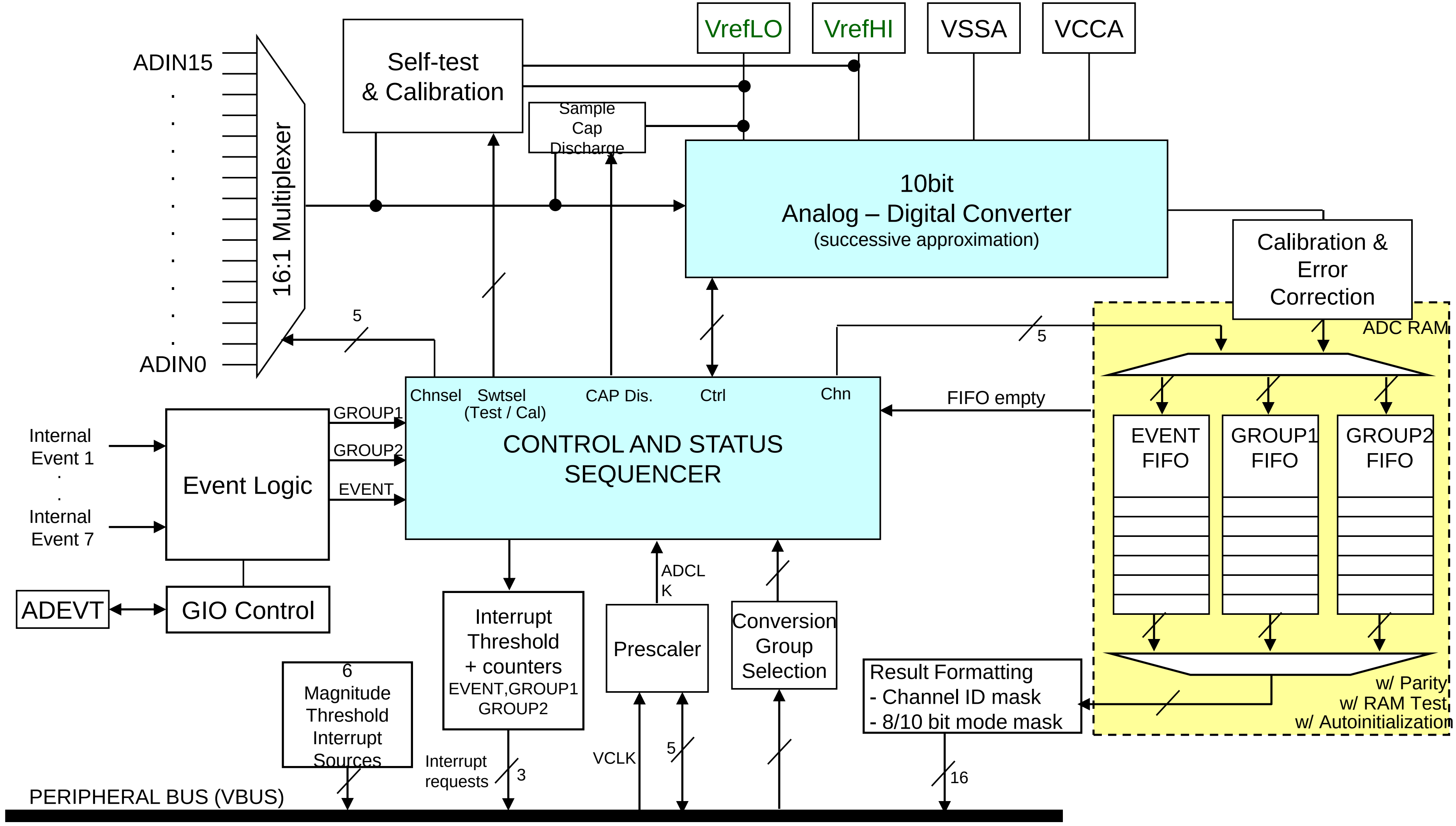
- The coding segment for this project is now complete, go ahead and build your project and then program it to the flash.

Testing your code

- Upon Completion open the TMS470M console or preferred terminal program.
- Ensure the following properties
  - Baud rate: 19200
  - Data bits: 8
  - No parity, 1 Stop bits
- Click the ‘Terminate All’ box in CCS then hit reset on the board.
- You should now see the ‘SCI UART IS SET.’ prompt in the console window.
- When character ‘1’ is typed, the microcontroller will output ‘‘ADC demo starts” to the terminal program. If other character is typed, the microcontroller will output “SPI demo starts”.

TMS470M: Multi-Buffered ADC (MibADC)

MibADC Block Diagram



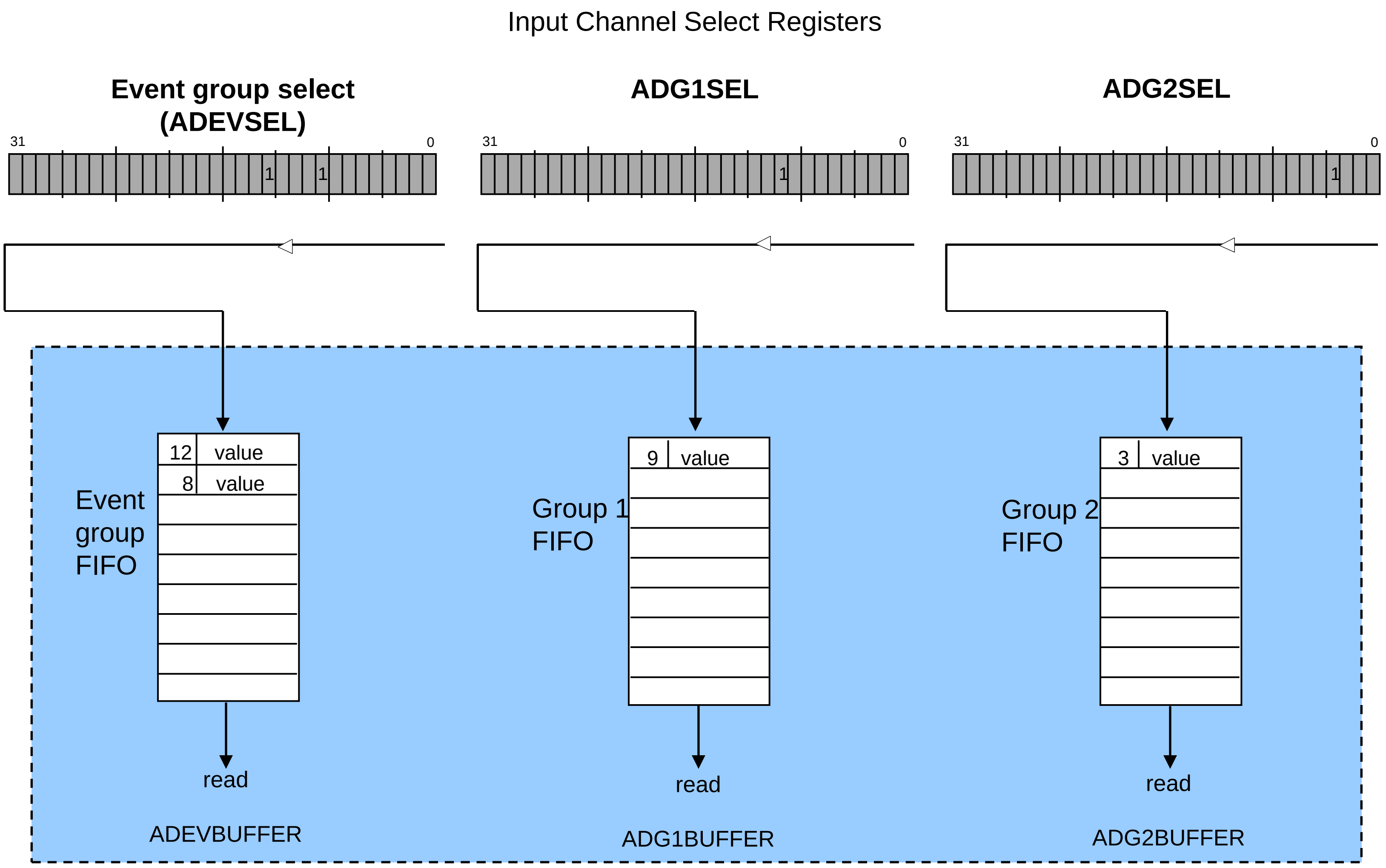
MibADC Main Features

- Configurable 10-bit or 8-bit resolution
- 16 channels
- Sequential multi-channel conversion in ascending order
- Two conversion modes
  - Single conversion
  - Continuous conversion
- Three conversion groups w/ programmable sample and acquisition times
  - Two software- or event-triggered conversion groups: Group1 and Group2
  - One event-triggered-only conversion group: Event Group
- Three size adjustable memory regions
  - Channel identifier stored with conversion result
- Up to 8 event trigger options
- Enhanced interrupt capability w/ programmable interrupt threshold counter
- Power-down mode
- Embedded self-test & calibration
- External event pin (ADEVT) can be programmed as general-purpose I/O

MibADC Operation Modes

- Conversion mode
  - normal active mode for converting the selected external input voltage
- Sample Capacitor Discharge mode
  - active mode that grounds the ADC sampling capacitor
- Calibration mode
  - special active mode for calibration using internal reference voltages
- Self-test mode
  - active mode for failure-detection using internal reference voltages
- Power-down mode
  - inactive mode in which the ADC internal clock is stopped

MibADC Conversion Groups



MibADC Interrupts

- Conversion Group End Interrupt
  - All channels that are assigned to a particular group are converted
- Conversion Group Buffers Threshold Interrupt
  - Number of conversion results exceed threshold register value
- Conversion Group Buffers Overrun Interrupt
  - Number of ADC conversions exceed the number of buffers allocated for that conversion group
- Magnitude Threshold Interrupt
  - Magnitude comparison of conversion result on up to six channels; alternately, comparison can be made between the conversion result from another channel.
- Parity Error Interrupt
  - Parity error following a read from the ADC RAM

TMS470M Support Structure

TMS470M Support Structure

TMS470 Web Page: www.ti.com/TMS470M

- Data Sheets
- Technical Reference Manual
- Application Notes
- Software & Tools Downloads and Updates
- Order Evaluation and Development Kits

TMS470M Engineer 2 Engineer Forums:

www.ti.com/hercules-support

- News and Announcements
- Useful Links
- Ask Technical Questions
- Search for Technical Content

TI E2E™ Community

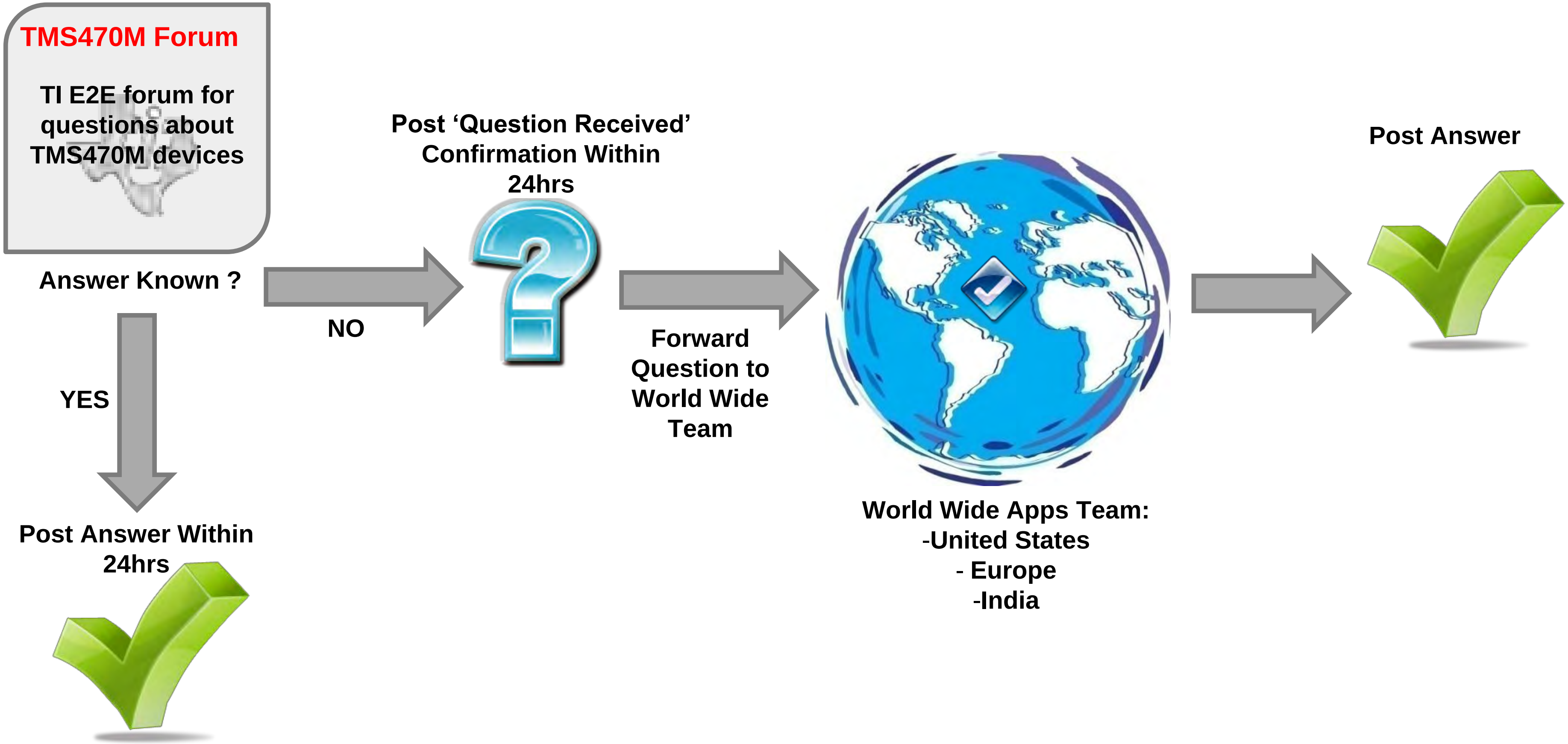
TMS470M WIKI:

www.ti.com/hercules-tms470m-wiki

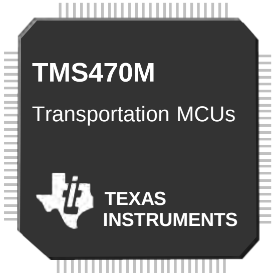
- How to guides
- Intro Videos
- General Information

TMS470M Microcontroller Forum Overview

Forum Flow:


TMS470M Forum
TI E2E forum for questions about TMS470M devices
Answer Known ?
NO
YES
Post 'Question Received' Confirmation Within 24hrs
Forward Question to World Wide Team
World Wide Apps Team:
-United States
- Europe
-India
Post Answer
Post Answer Within 24hrs

- Forum Guidelines:
  - At least one person will monitor the forum at all times (work days)
  - All questions posted in the forum will have a response in 24hrs or less



Thank You!

Please fill out the TMS470M 1 Day Training Class Survey

Texas Instruments