UNIVERSITY OF
LEICESTER

Department of Engineering

EG3205 Programming Microelectronic and Multi-Core Systems

Part II. Multiprocessor and Multicore Systems

Prof. Tanya Vladimirova
MEng MSc PhD CEng FIET SMIEEE MACM FHEA
Email: tv29@le.ac.uk

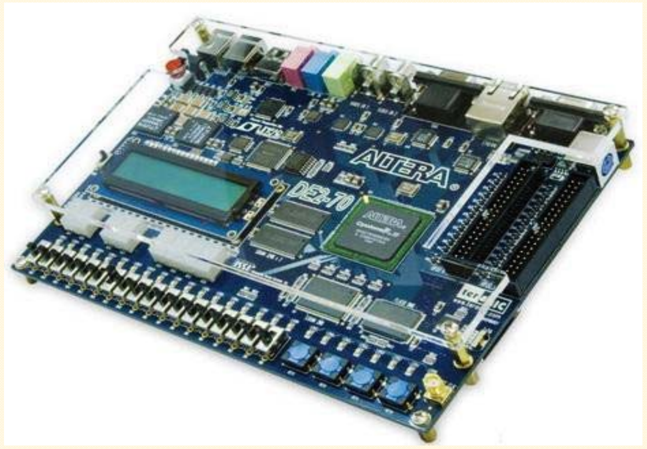
Outline

- Overview of the Lab 4 Exercise
- Controller Area Network (CAN)
- Shared-Clock Scheduling
- Lab 5: Dual-core Nios II Design

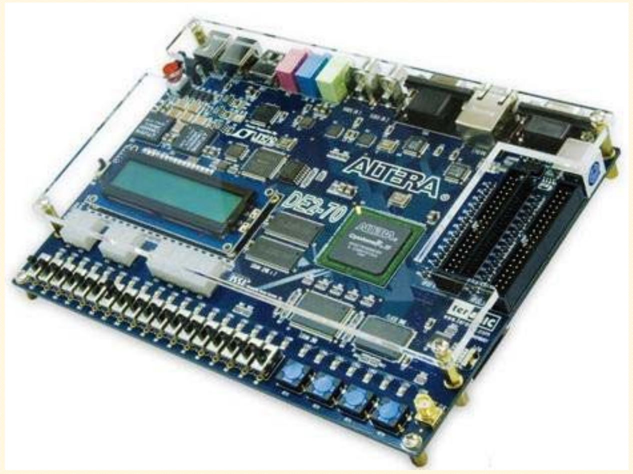
Lab 4 Exercise

Dual-Processor System Design Using CAN and TTC-SC Scheduler

Lab 3 Setup: Flexible implementation platform



+



Master:
FPGA

Slave:
FPGA

Lab 4 Exercise

Software provided:

- Master and slave software applications:
  - `app_master.zip`
  - `app_slave.zip`
- An SPI CAN driver communicating with the MCP2515 CAN Controller:
  - `SPICAN_Driver.zip`

Quartus II Reference Materials:

- Introduction to the Altera Qsys Tool
- Nios II Hardware Development Tutorial
- Embedded Design Handbook

Quartus II Reference Materials:

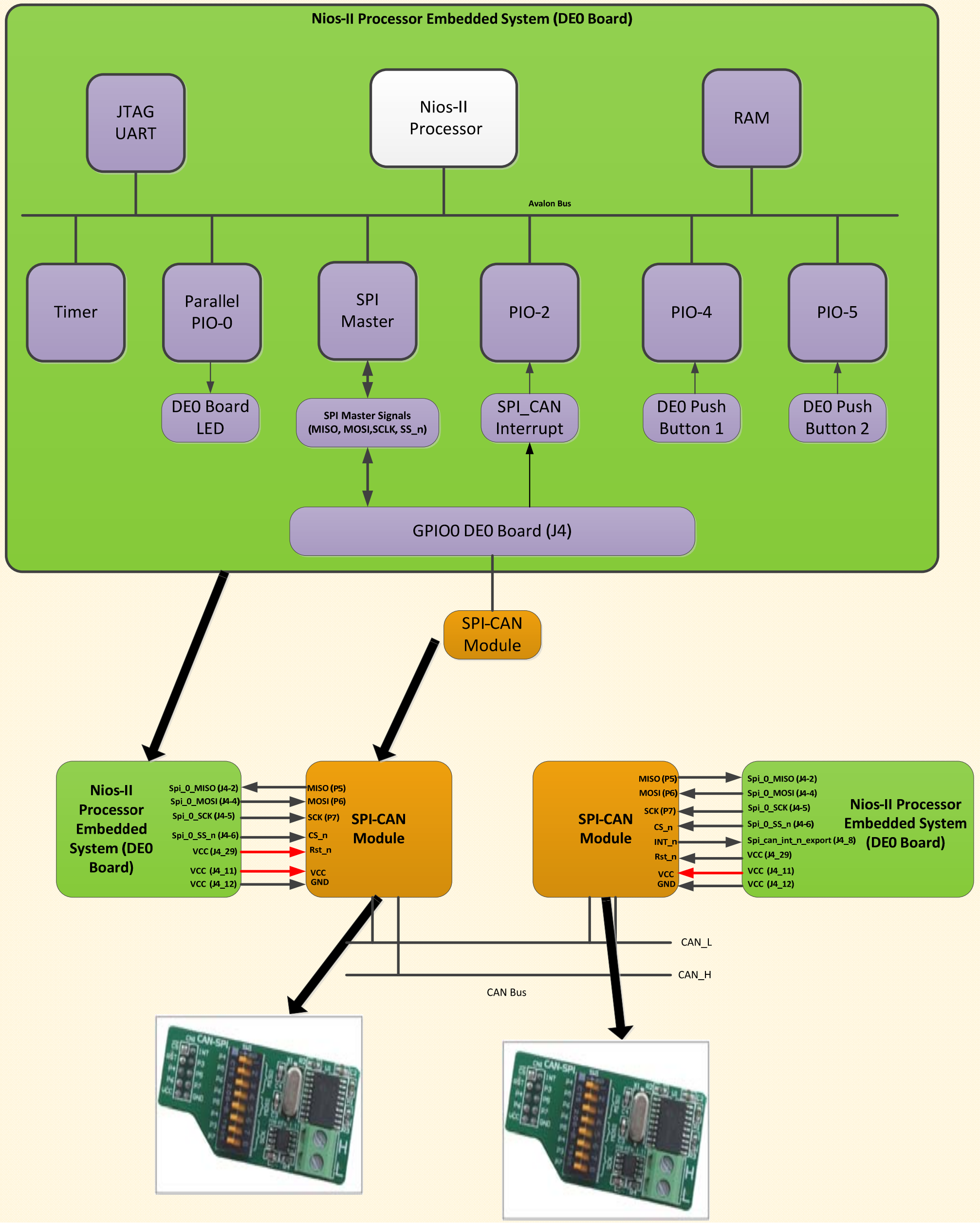
- DE0 board user manual
- CAN-SPI board manual

Scheduler Reference Materials:

- M Pont PTTES book

CAN / SPI Reference materials:

- SPI Spec
- Introduction to CAN
- MCP2515 CAN Controller IC



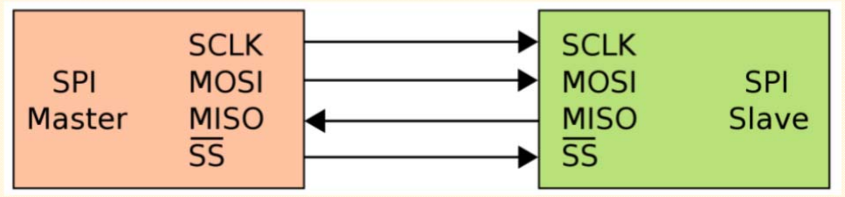
JTAG-UART IP Core

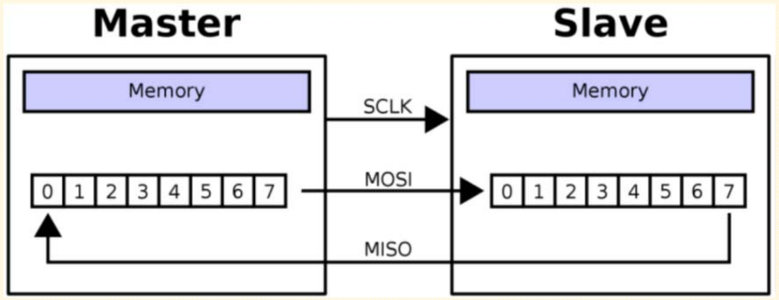
- An important difference between discrete microprocessors and FPGAs is that an FPGA contains no logic when it powers up.
- To enable your FPGA-based embedded system to behave as a discrete microprocessor-based system, your system should include JTAG[1] interface.
- The JTAG interface supports FPGA configuration and hardware and software debugging.
- You can perform the following tasks using the JTAG interface:
    - ➢Configure the FPGA
    - ➢Download and debug software
    - ➢Communicate with the FPGA through a UART[2]-like interface (JTAG UART)
    - ➢Debug hardware (with the SignalTap® II embedded logic analyzer)
    - ➢Program flash memory

[1]JTAG - Joint Test Action Group
[2]UART – Universal Asynchronous Receiver-Transmitter

Embedded Design Handbook, Altera, July 2011

SPI bus: Data Transmission





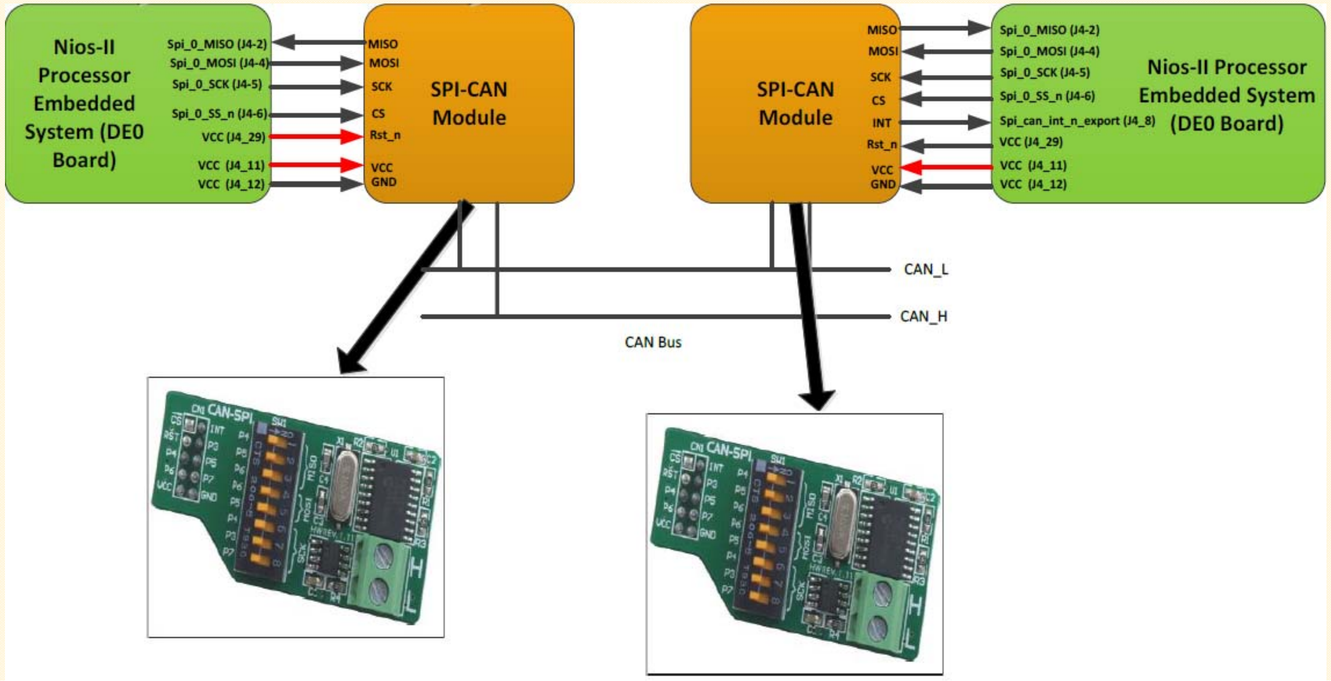
The SPI bus specifies four logic signals:

SCLK: serial clock (output from Master);

MOSI: Master output, Slave input (output from Master);

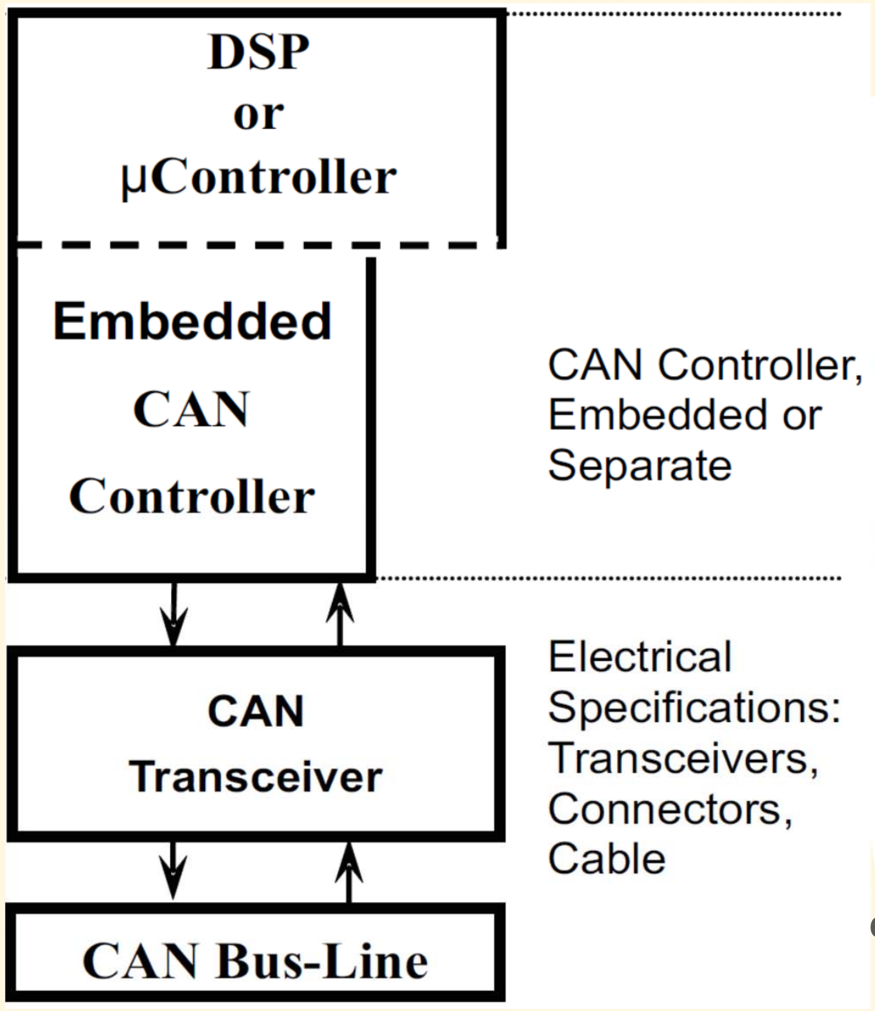
MISO: Master input, Slave output (output from Slave);

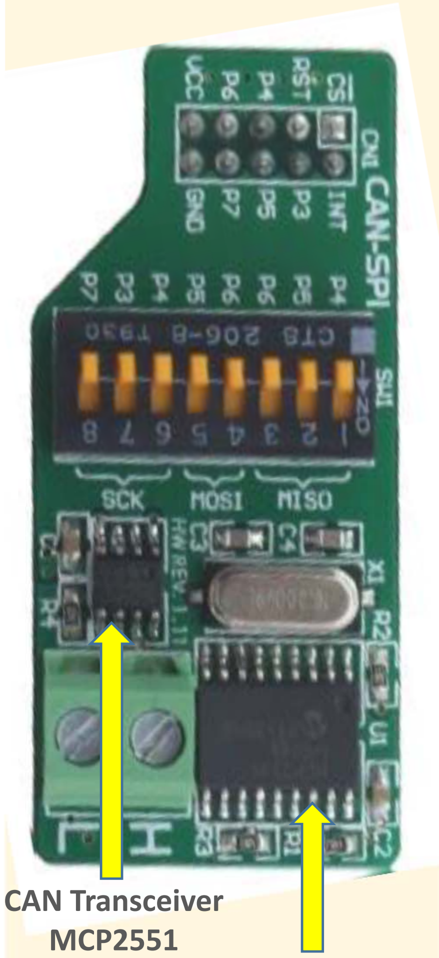
$\overline{SS}$: Slave select (active low, output from Master).



CAN

- CAN is a serial communications protocol defined by the International Standardization Organization (ISO)
  - ➢ originally developed for the automotive industry
  - ➢ to replace the complex wiring harness with a two-wire bus.
- The specification calls for
  - ➢ high immunity to electrical interference and
  - ➢ the ability to self-diagnose and repair data errors.
- These features have led to CAN's popularity in a variety of industries including building automation, medical, manufacturing, spacecraft, etc..


DSP
or
μController
Embedded
CAN
Controller
CAN Controller, Embedded or Separate
CAN
Transceiver
Electrical Specifications: Transceivers, Connectors, Cable
CAN Bus-Line

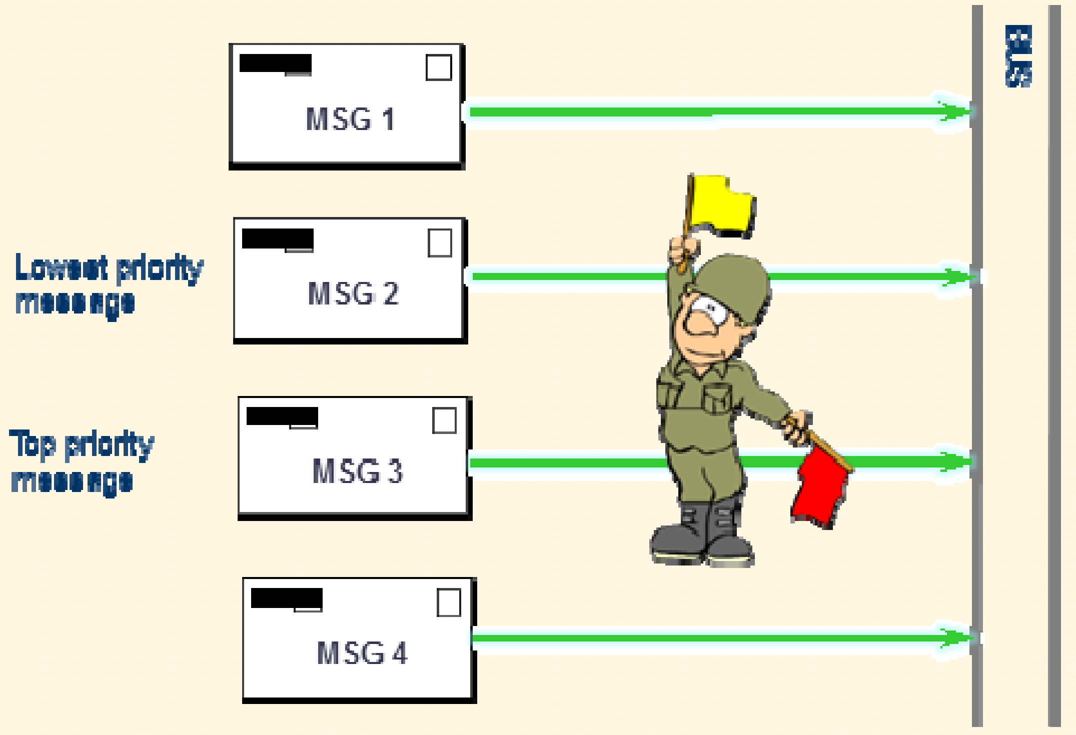


Introduction to the Controller Area Network (CAN), TI, 2008

Controller Area Network (CAN):

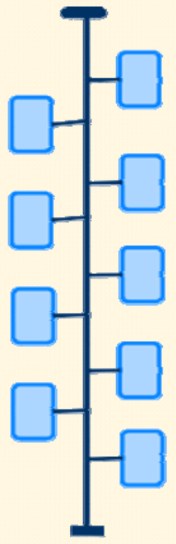
Data Frame Format

CAN Network protocol: CSMA/CA (Priority)



Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) using priorities:

- Node wishing to transmit must listen to the bus to check if the bus is free.
- If bus is “idle”, then message with the highest priority is transmitted.
- If bus is busy, then wait until bus is free.
- When a message is already transmitting, it has to finish its transmission before another message can be transmitted.
- Example: Controller Area Network



Adapted from course material developed by Prof. Michael J. Pont

CAN Features

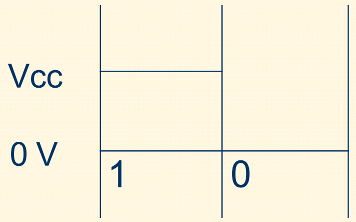
- Maximum number of nodes on a CAN Bus
  - Practical limits are between 32 to 64[1] nodes
  - Depends upon data rate and bus length
- Operates up to a 1 Mbps (depends on bus length).
  - 40 m at 1 Mbps
  - 1000 m at 40 Kbps
- Up to 8 bytes of data per message
- Low cost cabling
- CAN controllers are widely available
  - integrated in microcontrollers and “stand alone”

1: http://www.computer-solutions.co.uk/info/Embedded_tutorials/can_tutorial.htm

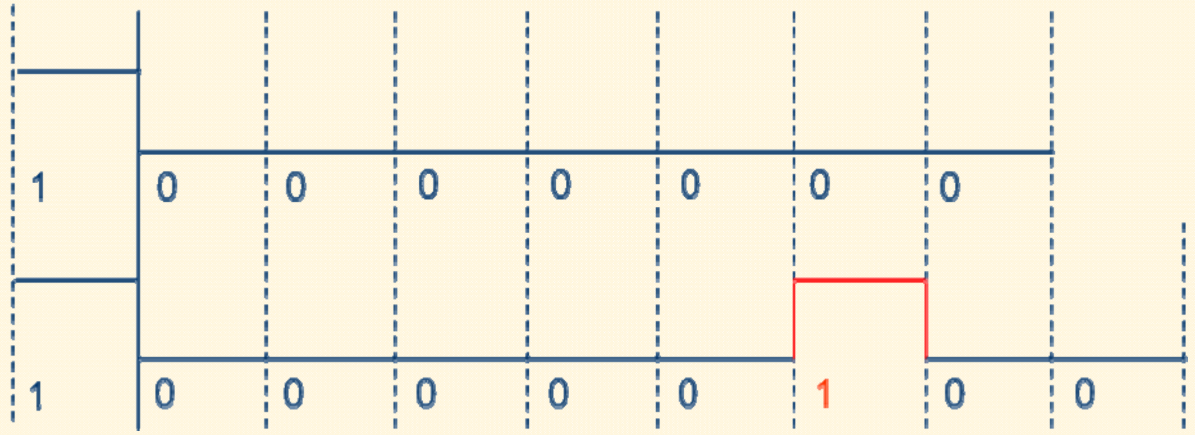
Adapted from course material developed by Prof. Michael J. Pont

Controller Area Network (CAN): Bit Coding

- CAN uses NRZ coding
  - high voltage → Logic ‘1’
  - low voltage → Logic ‘0’



CAN specifies two logical states:
recessive (Logic 1)
dominant (Logic 0)



Non-return to Zero (NRZ) method of encoding:
The signal remains at the binary level assigned to it for the entire bit time.
In other words, the voltage does not return to zero during the binary 1 interval.

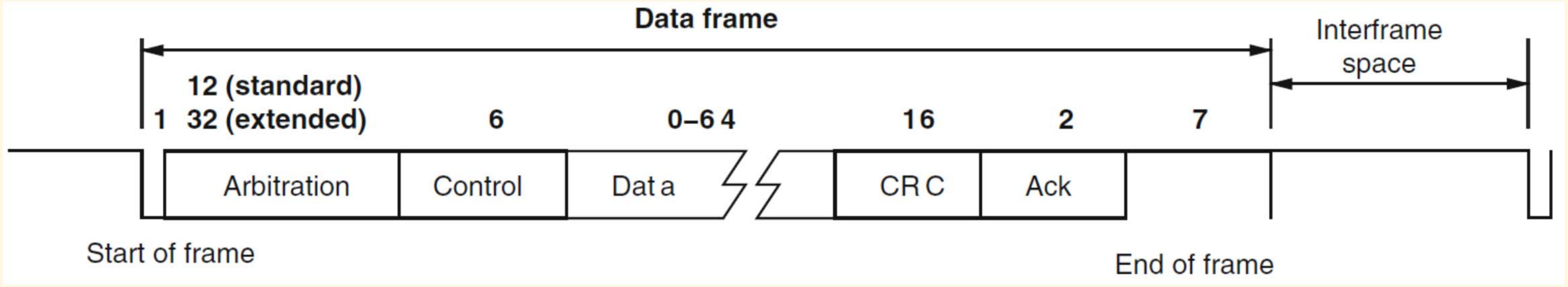
- CAN employs bit stuffing
  - Original frame: 00011111110011000001001111111 …
  - Sent frame: 000111110110011000001100111110 1 …

CAN Message Frames

- There are several CAN message formats
  - **Data Frame**
    - Carries a message from transmitter to receiver.
  - **Remote Frame**
    - Requests a Data Frame from another node.
    - The identifier used in remote and data frame will be same
  - **Error Frame**
    - Transmitted by any node that detects a bus error.
    - It has two parts an Error and an Error delimiter.
  - **Overload Frame**
    - When a node is not ready to receive any further data.
    - Notifies other nodes using overload frame.
    - Provides an extra delay between preceding and succeeding Data or Remote Frames
- We will focus on the **Data Frame** format as the most common message type.

Adapted from course material developed by Prof. Michael J. Pont

CAN Data Frame



- The CAN data frame format is shown above, where the size of the fields is expressed in bits.
- Each frame starts with a single dominant bit, interrupting the recessive state of the idle bus.
- The Data frame is composed of different fields.
- There are two types of Data Frame format:
  - Standard Data Frame
  - Extended Data Frame

Understanding and Using the Controller Area Network Communication Protocol Theory and Practice, Springer, 2012

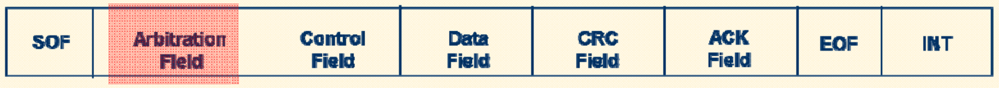
CAN Data Frame Format

SOF	Arbitration Field	Control Field	Data Field	CRC Field	ACK Field	EOF	INT

- The Data frame is composed of 7 different bit fields
- There is also an “inter-frame space” (**IFS**) between two data frames.
- The Start of Frame (**SOF**)
  - CAN bus is recessive (‘1’), when idle
  - **SOF** is a single dominant bit to signify the beginning of the Data Frame.
  - Also used to synchronise nodes on the CAN bus.
  - **SOF** can be sent immediately after **IFS** or on the third bit of **IFS**

Adapted from course material developed by Prof. Michael J. Pont

Arbitration Field

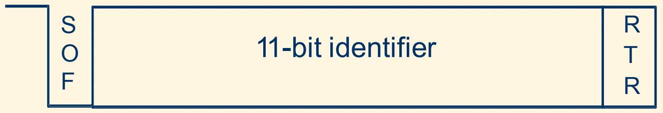


- The Arbitration field is different for a Standard Format and an Extended Format (Bosch, 1991).

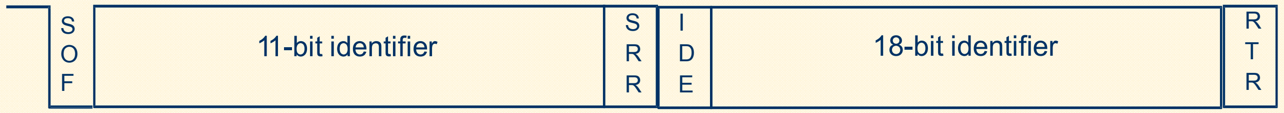
  – The Standard CAN **11-bit identifier** establishes the priority of the message. The lower the binary value, the higher its priority.

  – **RTR** (Remote Transmission Request) to distinguish:
    - a data frame (dominant) from
    - a remote frame (recessive).

Arbitration Field for a standard CAN Frame



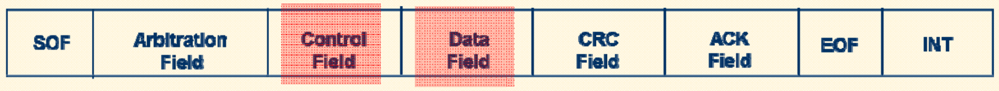
Arbitration Field for a extended CAN Frame



  – The **SRR** (Substitute Remote Request) bit has to be recessive (‘1’).
  – The **IDE** (Identifier Extension) bit has to be recessive (‘1’) indicating that more identifier bits follow.
  – The 18-bit extension follows **IDE**.
  – The **RTR** bit has to be dominant (‘0’).

- Arbitration field is also used for Filtering on the receiver side.

Adapted from course material developed by Prof. Michael J. Pont



Control and Data Fields

- The Control Field consist of **6** bits.

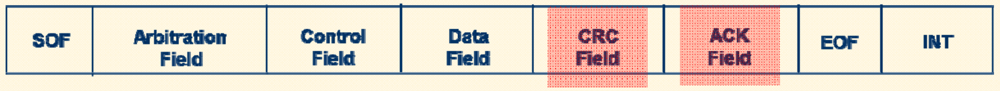
IDE/r1	r0	DLC3	DLC2	DLC1	DLC0

- For a standard message, a dominant (‘0’) **IDE** bit is transmitted.
- **r0** and **r1** are reserved bits.
- **DLC0-DLC3** specify the number of data bytes that this CAN message has (up to 8).

Num of bytes	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d

Num of bytes	DLC3	DLC2	DLC1	DLC0
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

Adapted from course material developed by Prof. Michael J. Pont

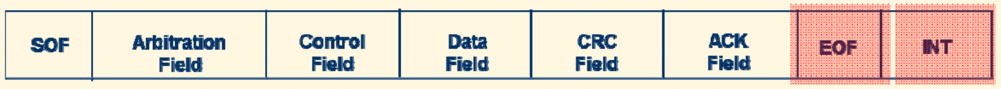


CRC and ACK Fields

- The 16-bit cyclic redundancy check (**CRC**) field is used for error detection and consists of:
  - 15 bits for the **CRC Sequence**, which represents the remainder of the polynomial division of the data contents and the generator:

    $x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$
  - followed by a single recessive (‘1) **CRC Delimiter** bit

- The **ACK** Field contains **2 bits**, each bit corresponding to an **ACK** (acknowledge) bit and **ACK Delimiter**.
  - The ACK Delimiter must always be recessive (‘1’).
  - The transmitter sets the ACK bit to be recessive (‘1’).
  - The receiver, upon receiving the matching CRC Sequence correctly, reports this in the ACK bit of the transmitter message by changing it to a dominant (‘0’) bit.



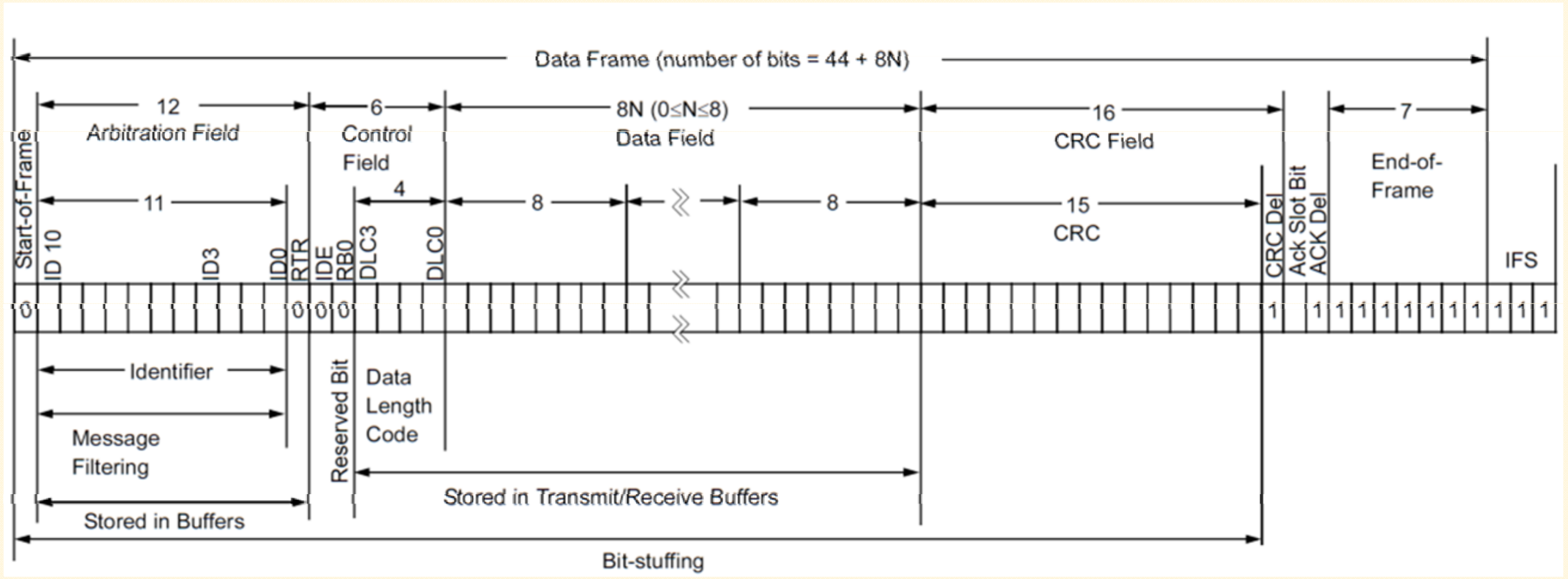
EOF and Interframe Space

- The **EOF** contains **7** recessive(‘1’) bits.
- The **EOF** is followed by **3** recessive(‘1’) bits to separate consecutive frames.
- These three bits are called inter frame space (**IFS**)

Adapted from course material developed by Prof. Michael J. Pont

Example: MCP2515 Standard Data Frame

The MCP2515 is a stand-alone CAN controller developed to simplify applications that require interfacing with a CAN bus.



RTR (Remote Transmission Request) - to distinguish a data frame (dominant) from a remote frame (recessive)
IDE (Identifier Extension) bit, must be dominant to specify a standard frame
RB0 (Reserved Bit Zero) - defined as a dominant bit by the CAN protocol
DLC (Data Length Code), which specifies the number of bytes of data
CRC (Cyclic Redundancy Check) field
CDR Del - recessive CRC Delimiter bit

ACK (Acknowledge) - two-bit field
During the **ACK Slot** bit, the transmitting node sends out a recessive bit. Any node that has received an error-free frame acknowledges the correct reception of the frame by sending back a dominant bit
ACK Del (acknowledge delimiter) - the recessive **ACK Del** completes the acknowledge field and may not be overwritten by a dominant bit.

MCP2515 Datasheet, p.37

Shared Clock Scheduling:

Overview

Time-Triggered Multiprocessor Systems

- An important advantage of time-triggered (co-operative) scheduling it is inherently scaleable and extends naturally to multiprocessor environments.
- Some of the challenges that face developers who wish to design multiprocessor applications:
  - How do we keep the clocks on the various nodes synchronized?
  - How do we transfer data between the various nodes?
  - How does one node check for errors on the other nodes?
- By using a shared-clock (S-C) scheduler, we can address all three problems.
- Moreover, the time division multiple access (TDMA) protocol we employ to achieve this is
  - a ‘natural extension’ to the time-triggered scheduling for single-processor systems which we have used in the EG3205 lab exercises so far.

M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014

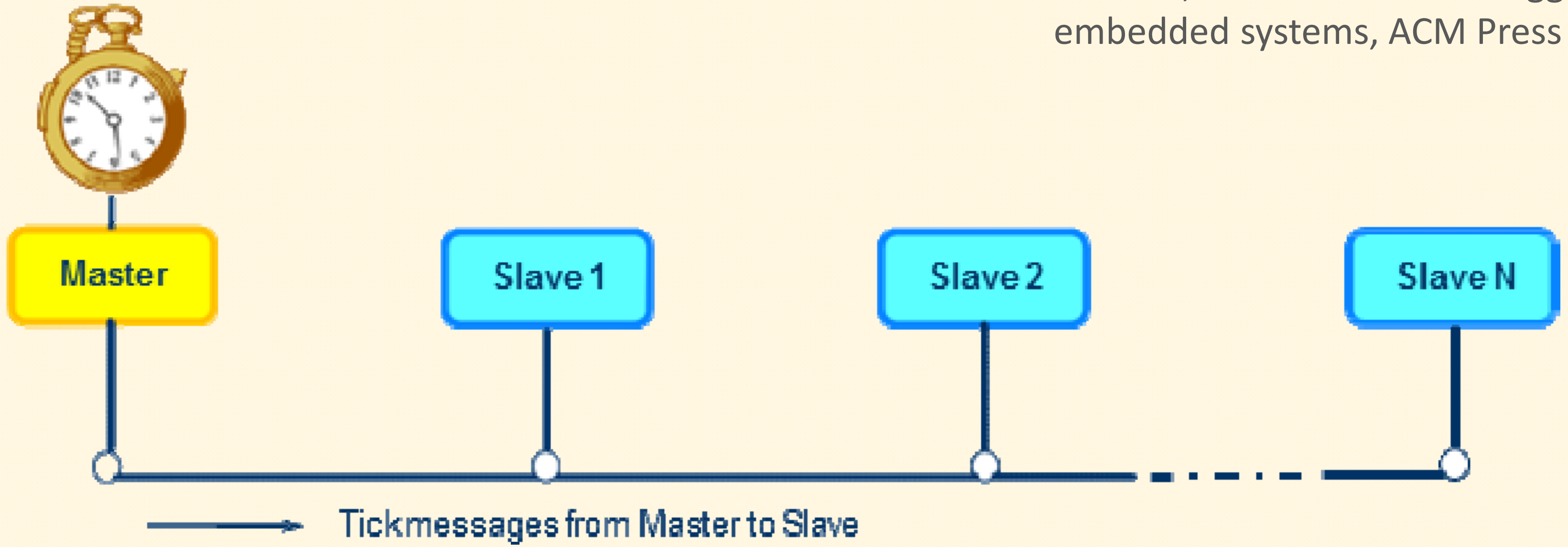
Important assumptions in a Shared-Clock network

- Shared-clock designs are based on a “time division multiple access” (TDMA) protocol.
- For correct operation, we assume that the communication medium (e.g. CAN network) is free when we wish to send either a Tick or Acknowledgement message.

Sharing a single clock: The simplest form of S-C scheduler

- A single accurate clock on the Master node is used to keep all the Slaves on the network in sync.
- The clock on the Master node is used to drive its scheduler, as in any single processor system.
- This involves the sending of tick messages from the Master to the Slave(s) at regular intervals.

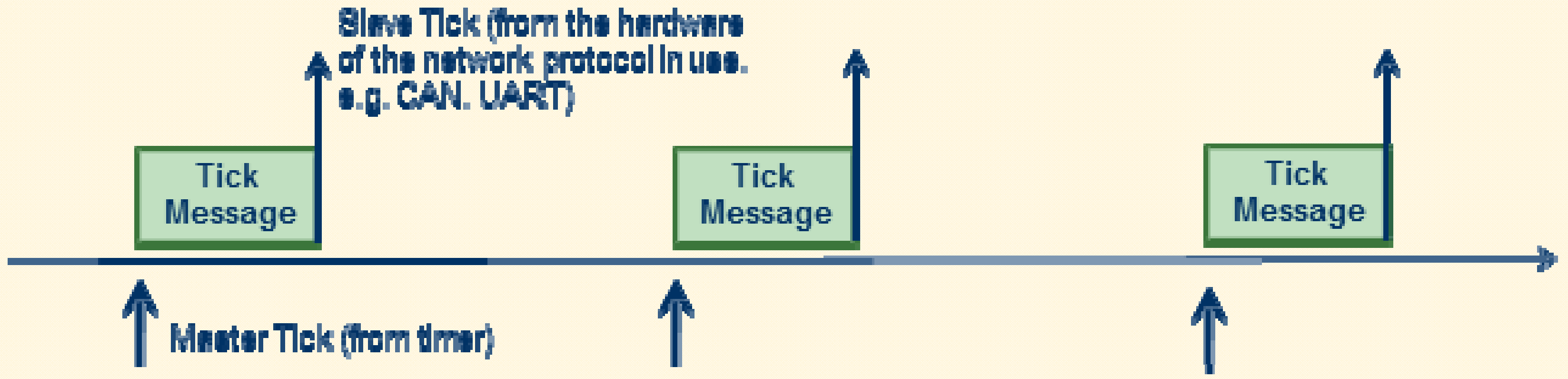
M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014



Sharing a single clock: The simplest form of S-C scheduler

- The Slaves also have a scheduler.
- The interrupts used to drive this scheduler are derived from the “Tick” messages generated by the Master.
- Thus, in a CAN-based network,
  - the Slave node will have a S-C scheduler driven by the ‘receive’ interrupts generated through the receipt of a byte of data sent by the Master.

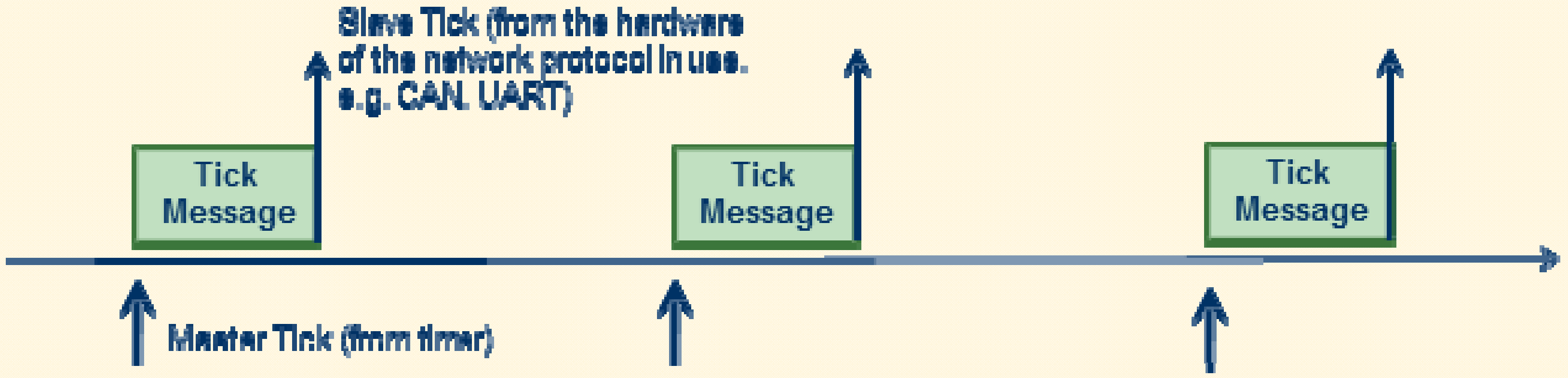
M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014


Slave Tick (from the hardware of the network protocol in use. e.g. CAN. UART)
Tick Message
Tick Message
Tick Message
Master Tick (from timer)

Communication between the Master and Slave nodes in a CAN-based S-C network

- This involves the transmission of the tick messages, triggered by the timer ticks in the Master.
- The receipt of the regular messages by the Slave is the source of the interrupt used to drive the scheduler in this node:
  - ➢ the Slave does not have a separate timer-based interrupt.
- By default, there is a slight delay between the tick generation on the Master and Slave nodes in some networks.
  - ➢ This delay is short (a fraction of a millisecond in most cases); equally important, it is predictable and fixed.

M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014


Slave Tick (from the hardware of the network protocol in use. e.g. CAN, UART)
Tick Message
Tick Message
Tick Message
Master Tick (from timer)

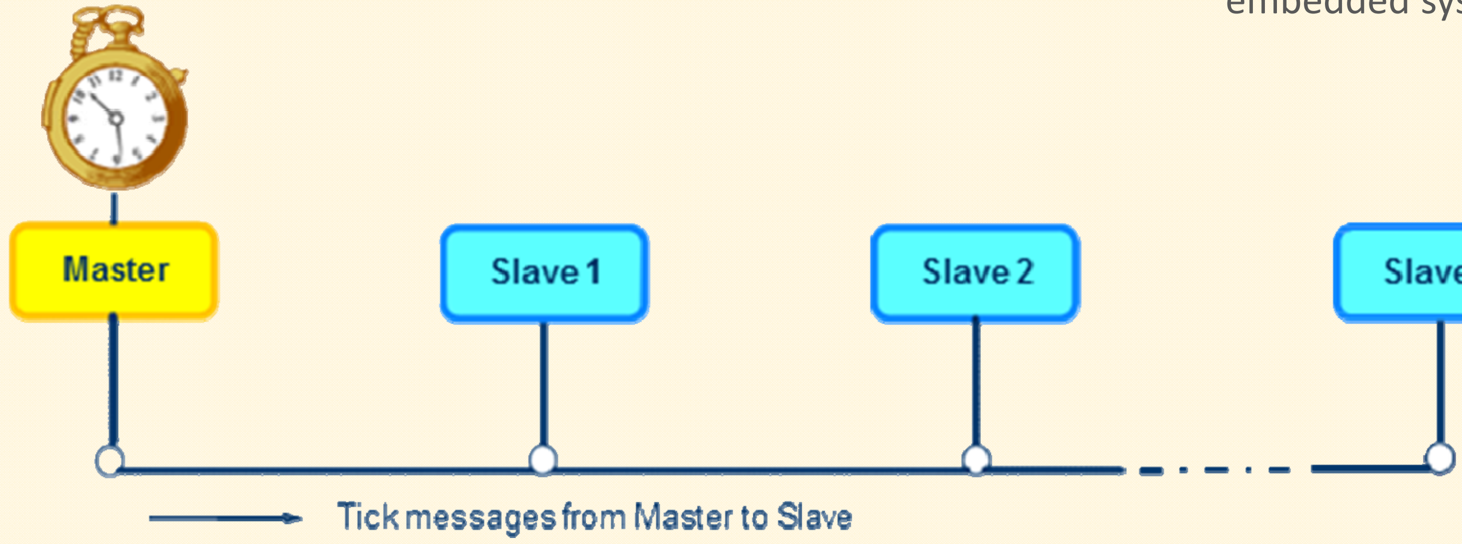
Transferring data

- To support the transfer of data from the Slave to the Master, we need an additional mechanism:
  - ➢this is provided through the use of ‘acknowledgement’ messages.
- The end result is a simple and predictable ‘time division multiple access’ (TDMA) protocol, in which acknowledgement messages are interleaved with the tick messages.

Data Transfer (Master to Slave)

- The Master node sends Tick messages regularly to all Slaves (for instance once every 10 millisecond).
- These Ticks, in most Shared-Clock Architectures, can include data transfer.

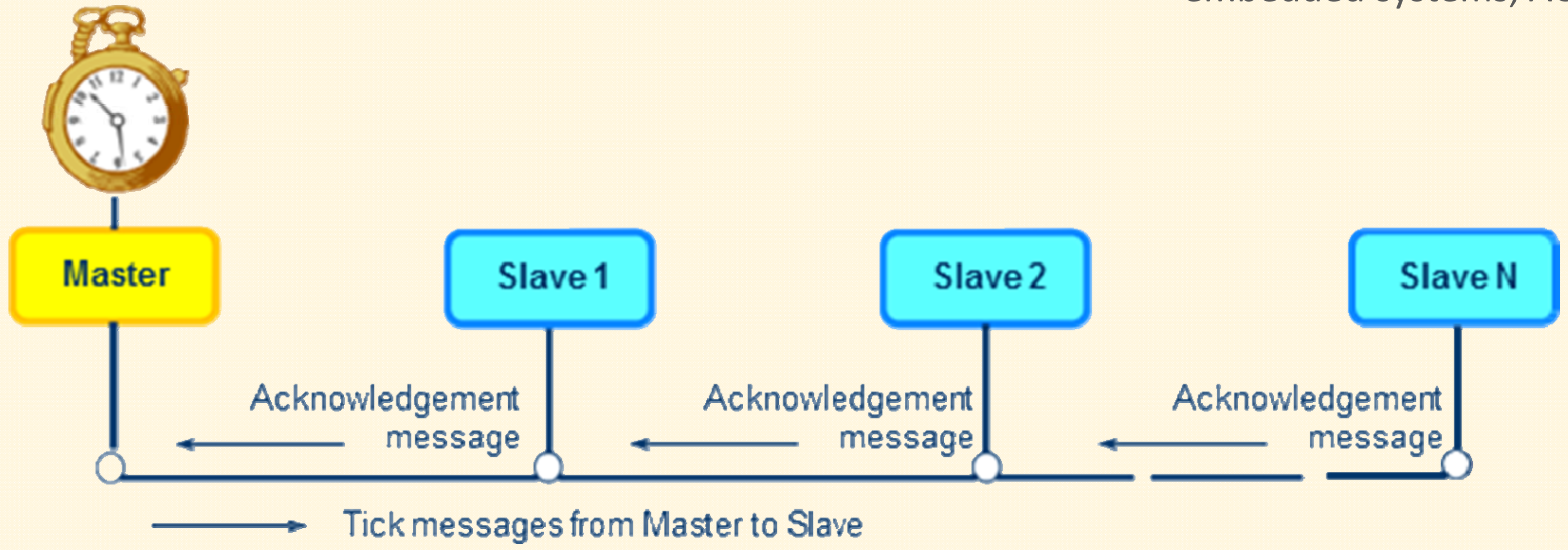
M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014



Data Transfer (Slave to Master)

- To deal with data transfer from the Slave to the Master, an additional mechanism is required.
- This is provided through the use of an acknowledgement message.

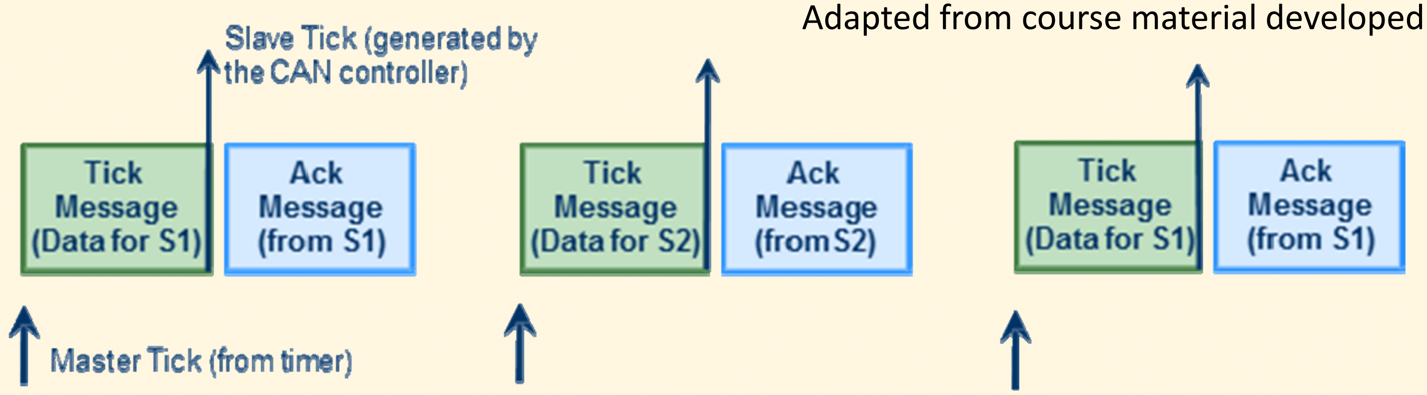
M. Pont, Patterns for time-triggered embedded systems, ACM Press Books, 2014



Acknowledgement Message Structure

- All Slaves generate interrupt in response to <u>every</u> Tick message.
- Only the Slave to which a Tick message is addressed will reply to the Master; this reply takes the form as an Acknowledgement message.
- Each Acknowledgement message from a Slave is between one and eight bytes long:
  - ➢ All of the bytes are sent in the Tick interval in which the Tick message was received.
  - ➢ The first byte of the Acknowledgement message is the ID of the Slave from which the message was sent; the remaining bytes (if any) are the message data.

Adapted from course material developed by Prof. Michael J. Pont

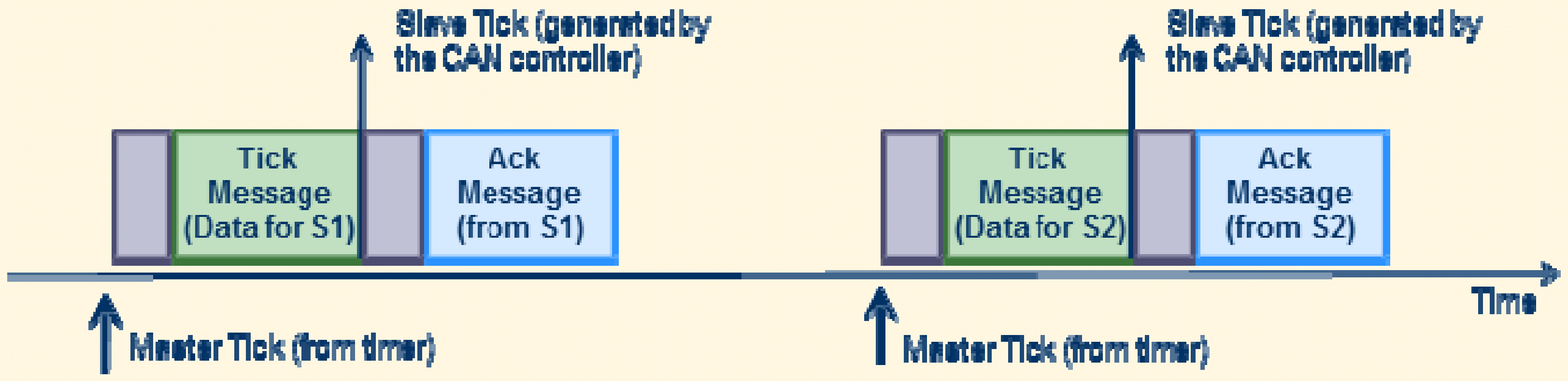

Slave Tick (generated by the CAN controller)
Tick Message (Data for S1)
Ack Message (from S1)
Tick Message (Data for S2)
Ack Message (from S2)
Tick Message (Data for S1)
Ack Message (from S1)
Master Tick (from timer)

Shared-Clock CAN Scheduler

- The scheduler “Tick” on the Master node is triggered by a timer.
- The scheduler “Tick” on the Slave node is triggered by a CAN message receive interrupt.

Message processing for SC-CAN (Master)

- Gets ACK message from **previous** Slave node.
- Sends Tick message with the current Slave ID.
- Check for errors
- The task dispatcher is then invoked.



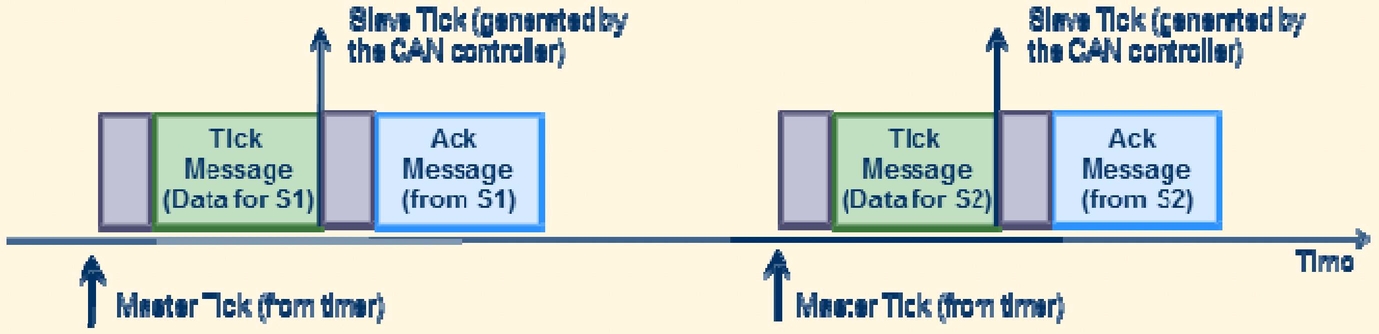
Slave Tick (generated by the CAN controller)
Tick Message (Data for S1)
Ack Message (from S1)
Master Tick (from timer)
Slave Tick (generated by the CAN controller)
Tick Message (Data for S2)
Ack Message (from S2)
Master Tick (from timer)
Time

Adapted from course material developed by Prof. Michael J. Pont

Message processing for SC-CAN (Slave)

- Checks if the Tick message from the Master matches the Slave ID for that particular slave.
- If the ID match, then an Acknowledgement message is sent back to the Master with the ID of the Slave in its message.
- If the ID does not match, then the Slave does not send an Acknowledgement to the Master.
- The task dispatcher is then invoked.

Adapted from course material developed by Prof. Michael J. Pont


Slave Tick (generated by the CAN controller)
Tick Message (Data for S1)
Ack Message (from S1)
Master Tick (from timer)
Slave Tick (generated by the CAN controller)
Tick Message (Data for S2)
Ack Message (from S2)
Master Tick (from timer)
Time

Hand-shake

- Purpose of the handshake:
  - To establish communication with each Slave in the system.
  - Deal with missing slaves (if necessary).
  - Start the slaves up.

- The shared-clocked CAN architecture keeps all the nodes on the bus synchronised by using a common clock.
- One important aspect of the synchronisation is the initial "hand- shake" between the Master and all the Slave nodes.

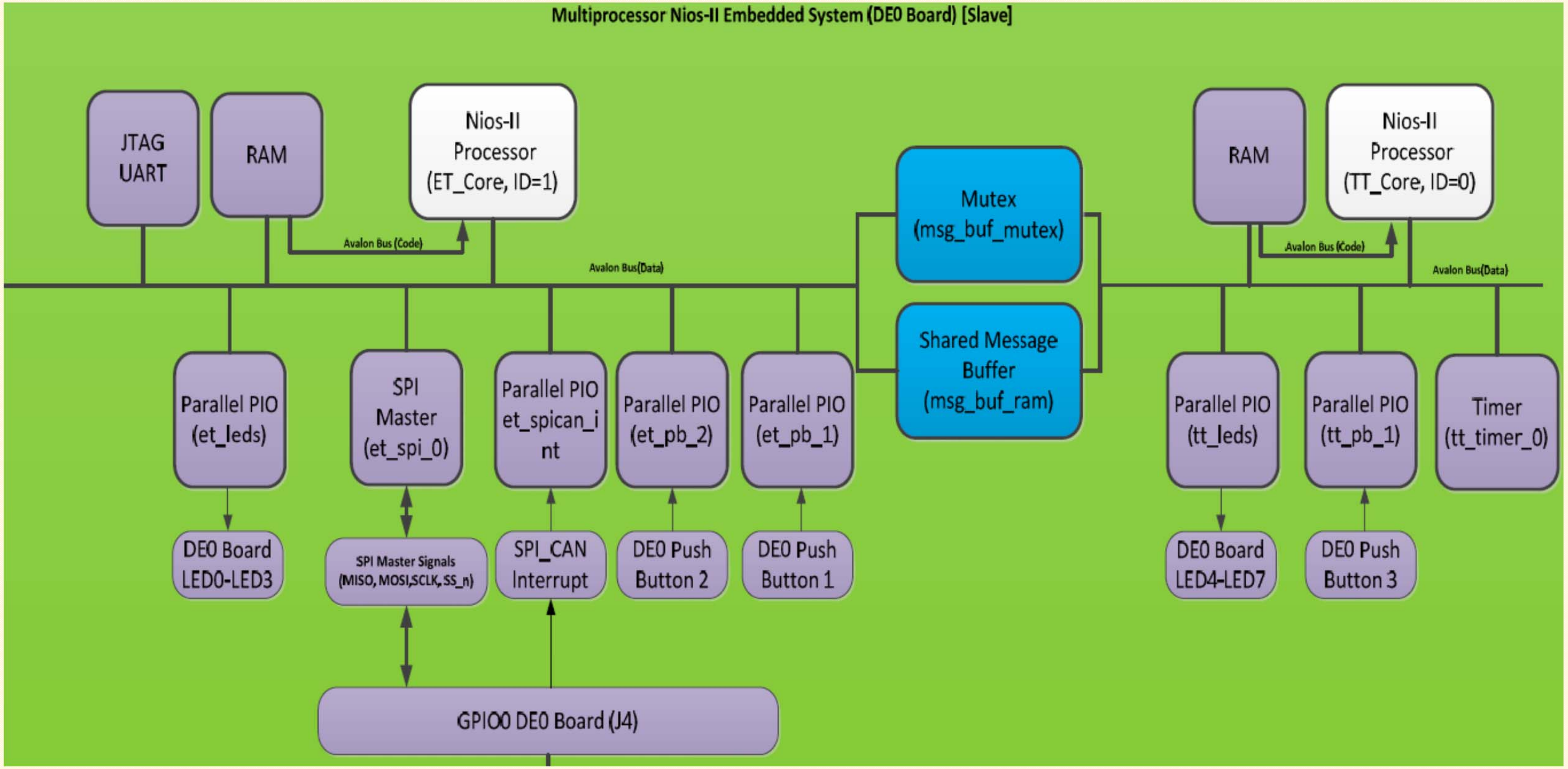
Adapted from course material developed by Prof. Michael J. Pont

Implementation of the “handshake”

- Master sends the following:
  - Data Byte 0 = 0x00
  - Data Byte 1= Slave ID
- The Slave with that particular ID should then reply the same
  - Data Byte 0 = 0x00
  - Data Byte 1 = Slave ID
- If Master receives the correct “code”, then it has successfully established contact with that Slave.
- The Master then tries to contact the next Slave on the bus.

Adapted from course material developed by Prof. Michael J. Pont

Lab 5: Dual-core Nios II Design


Multiprocessor Nios-II Embedded System (DE0 Board) [Slave]
JTAG UART
RAM
Nios-II Processor (ET_Core, ID=1)
Avalon Bus (Code)
Avalon Bus(Data)
Mutex (msg_buf_mutex)
Shared Message Buffer (msg_buf_ram)
RAM
Nios-II Processor (TT_Core, ID=0)
Avalon Bus (Code)
Avalon Bus(Data)
Parallel PIO (et_leds)
SPI Master (et_spi_0)
Parallel PIO et_spican_int
Parallel PIO (et_pb_2)
Parallel PIO (et_pb_1)
Parallel PIO (tt_leds)
Parallel PIO (tt_pb_1)
Timer (tt_timer_0)
DE0 Board LED0-LED3
SPI Master Signals (MISO, MOSI,SCLK, SS_n)
SPI_CAN Interrupt
DE0 Push Button 2
DE0 Push Button 1
DE0 Board LED4-LED7
DE0 Push Button 3
GPIO0 DE0 Board (J4)