Programming Intelligent Physical Systems

Lecture 2

Samarjit Chakraborty

Technical University of Munich
Chair of Real-Time Computer Systems (RCS)

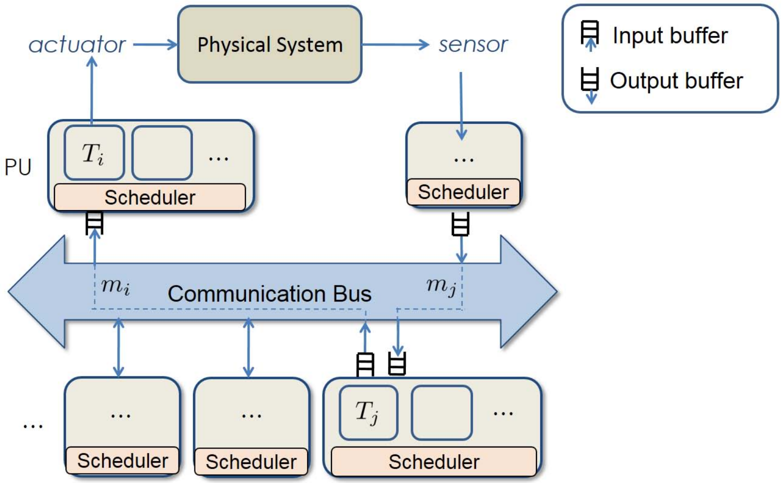
04 August, 2019

Outline of this lecture

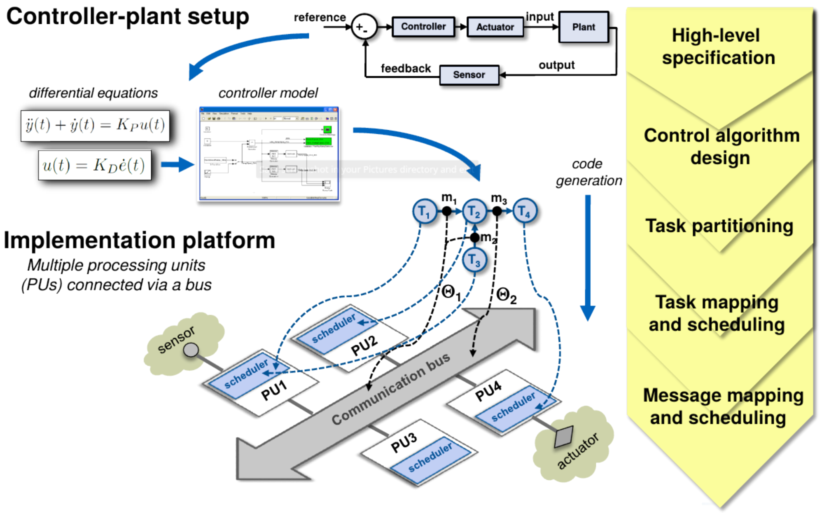
In today's lecture we will discuss:

- Distributed cyber-physical system architectures
- Communication in cyber-physical systems
- Introduction to different communication protocols like TDMA, CAN and FlexRay

Distributed/networked embedded systems

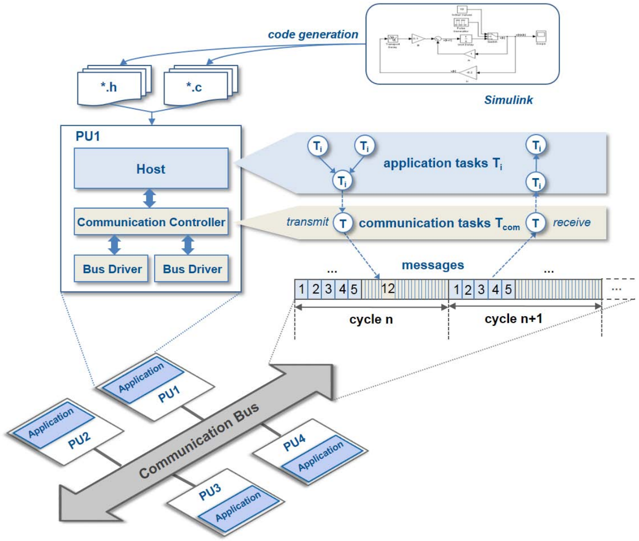


Design flow


Controller-plant setup
reference
+
-
Controller
Actuator
input
Plant
feedback
Sensor
output
differential equations
$\ddot{y}(t) + \dot{y}(t) = K_P u(t)$
$u(t) = K_D \dot{e}(t)$
controller model
code generation
Implementation platform
Multiple processing units (PUs) connected via a bus
T_1
T_2
T_3
T_4
m_1
m_2
m_3
Θ_1
Θ_2
sensor
scheduler
PU1
PU2
PU3
PU4
Communication bus
actuator
High-level specification
Control algorithm design
Task partitioning
Task mapping and scheduling
Message mapping and scheduling

System Description

Distributed embedded controller: implementation

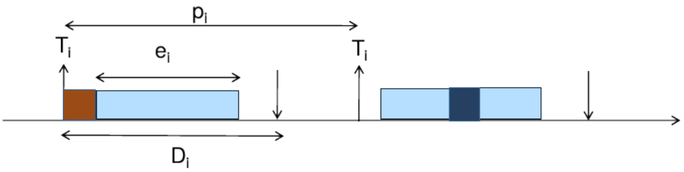


Processing Units (PUs)

- Host micro-controller
  - It runs real-time operating system or scheduler which schedules the application task T_i and the communication task T_{com}
  - The tasks interact with the physical world using sensors and actuators.
  - Sensors and actuators – connected to the processors via dedicated communication link (e.g., LIN in automotive, Profinet and Ethernet in industrial automation)
- Communication controller
  - It implements the bus protocol such as CAN, FlexRay.
  - It facilitates tracking of all network activities.
  - It allows buffering of incoming and outgoing data.
- Bus driver
  - Converts the bit streams into physical signals propagated over the bus.
  - Acts as an physical interface between the communication controller and the bus medium.

Application Tasks

- We consider a periodic preemptive scheduling policy implemented by scheduler or RTOS.
- The scheduler or RTOS provides a task dispatcher, which allows cyclic task execution.
- A dispatch event for a task can be defined as $T_i : p_i, D_i, e_i$

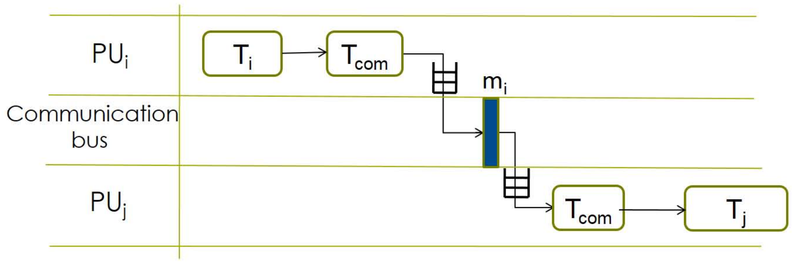


- ✓ Period p_i
- ✓ Relative deadline D_i
- ✓ WCET e_i

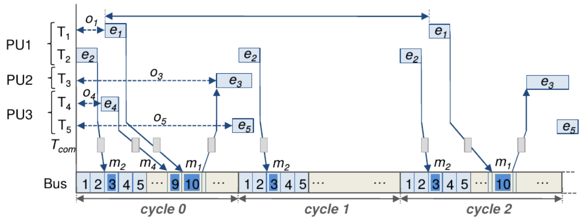
Communication Tasks

- One or multiple tasks communicate by exchanging messages over the communication bus
- The outcome of a set of (sending) tasks mapped at PU_i are packetized as messages m_i and sent to other tasks mapped at another PU_j via the communication bus
- A communication task T_{com} writes the output message m_i of the sending application task T_i to the dedicated output buffers of the communication controller
- In the receiving PU, the communication task T_{com} reads the corresponding input buffer and forwards the unpacked data to the application task T_j for further processing
- Hence, every communication task generates dispatch events similar to application tasks and needs to be considered in the schedulability analysis

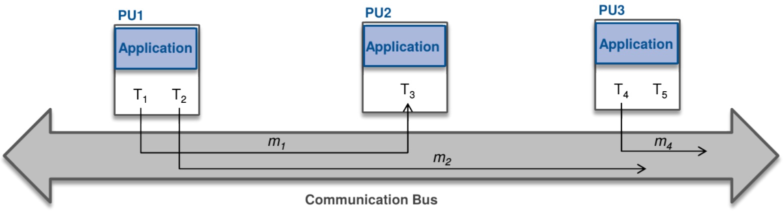
Communication over the bus



An Example



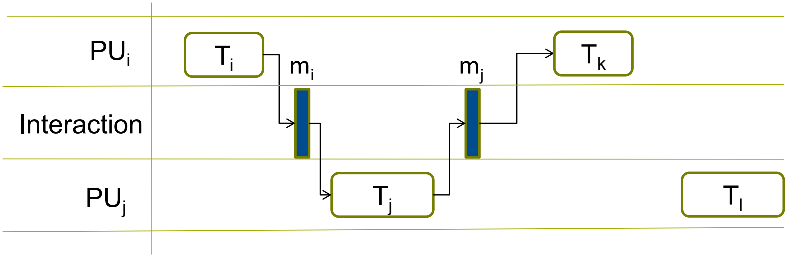
PU1
PU2
PU3
T1
T2
T3
T4
T5
Tcom
Bus
o1
o3
o4
o5
e1
e2
e3
e4
e5
m1
m2
m4
1 2 3 4 5 ··· 9 10 ···
cycle 0
cycle 1
cycle 2



PU1
PU2
PU3
Application
T1
T2
T3
T4
T5
m1
m2
m4
Communication Bus

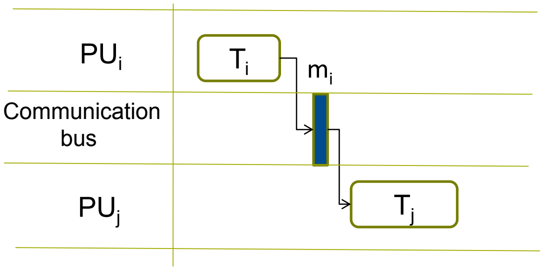
Communication over the Bus

Data Dependency



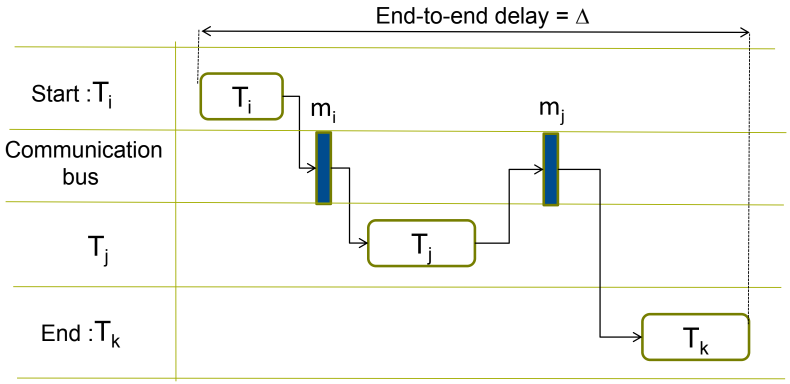
- One or multiple tasks process data generated by some other tasks.
- This causes data dependency among the tasks.

Messages over the communication bus



- In a distributed setting, the data dependency necessitates communication over the bus by exchanging messages
- Depending on the communication protocol, the messages are scheduled on the bus

End-to-end delay



- End-to-end path: $T_i \rightarrow$ bus $\rightarrow T_j$ bus $\rightarrow T_k$
- End-to-end delay is important to meet the application level requirements

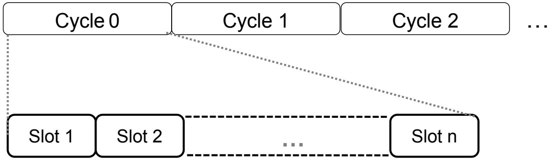
Communication bus

- Communication between various PUs is established by exchanging messages over the communication bus.
- Multiple messages are exchanged over the bus.
- The transmission of a message over the bus happens according to the bus communication protocol. E.g., Control-area-network – CAN, FlexRay, Ethernet.
- When multiple messages try to get transmitted over the bus simultaneously, the messages get transmitted according to the message schedule depending upon the bus protocol.
- Each message is assigned a message schedule which is calculated according to the bus protocol.
- The bus protocol can be
  - Time-triggered protocol– e.g., TDMA
  - Event-triggered protocol– e.g., CAN
  - Hybrid-protocol protocol– e.g., FlexRay

Time-triggered Communication

Time-triggered communication: TDMA

- Message transmission happens in pre-defined time-window
- Typical example is Time Division Multiplex Access (TDMA) where the messages are transmitted in a pre-defined time-windows called slots
- A TDMA communication is characterized by (i) TDMA cycle length (T) (ii) slot length (S)



- The slot length can be different for different slots, i.e., S_i for $1 \leq i \leq n$

TDMA: timing properties

- The slot length and the TDMA cycle length is related by:

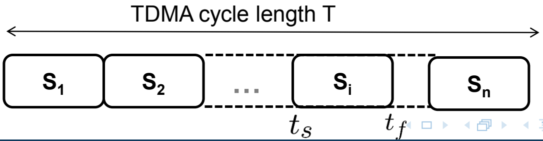
$$T = \sum_{i=1}^{n} S_i$$

- If the slot length is equal and there are n number of slots in the TDMA cycle:

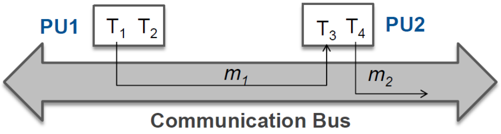
$$S = \frac{T}{n}$$

- If a message m_i is scheduled to slot S_i, the start time t_s and finish time t_f of message transmission are given by:

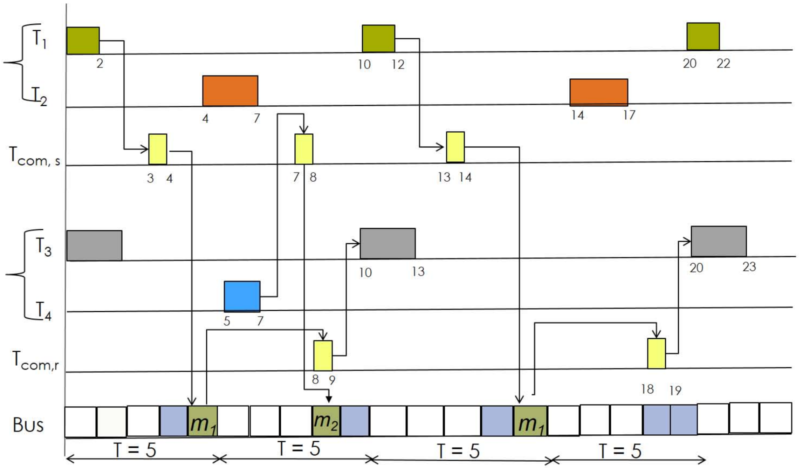
$$t_f = t_s + S_i$$



Time-triggered architecture: an example



- Time-triggered scheduling scheme is running on PU1 and PU2.
- Time-triggered task schedule is represented as $T_i \sim o_i, p_i, e_i$
- Task schedules are given as follows:
    - $T_1 : 0, 10, 2, \; T_2 : 4, 10, 3, \; T_3 : 0, 10, 3, \; T_4 : 5, 20, 2$
    - Communication tasks for m_1: $T_{com,s} = \{3, 10, 1\}$, $T_{com,r} = \{8, 10, 1\}$
    - Communication tasks for m_2: $T_{com,s} = \{7, 20, 1\}$
- The communication bus is running under TDMA with cycle length T = 5 and equal slot length S = 1. Assume that sizes of m_1 and m_2 fit into 1ms long slot length. That is, m_1 and m_2 can be sent using one slot
- Compute the end-to-end delay from $T_1 \rightarrow \text{bus} \rightarrow T_3$

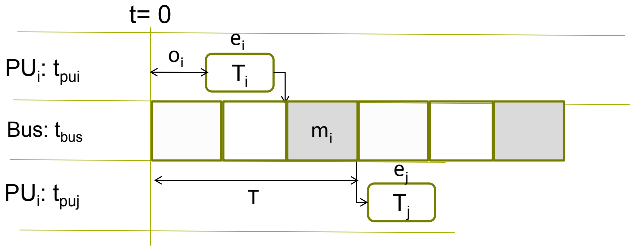

T_1
T_2
$T_{com,s}$
T_3
T_4
$T_{com,r}$
Bus
2
10
12
20
22
4
7
14
17
3
4
7
8
13
14
10
13
20
23
5
7
8
9
18
19
m_1
m_2
m_1
T = 5
T = 5
T = 5
T = 5

- m_1 and m_2 are assigned to slot 5 (S_5) and slot 4 (S_4) of TMDA communication cycle respectively.
- TMDA cycle length $T = 5$. m_1 is generated periodically with period being equal to its sending task T_1. Since $p_1 = 10$, a message m_1 is generated every 10 time unit. Therefore, in every alternate TDMA cycle, S_5 goes utilized
- m_2 is generated periodically with period being equal to its sending task T_4. Since $p_4 = 20$, a message m_2 is generated every 20 time unit. Therefore, every three out of four S_4 go utilized
- the end-to-end delay for path $T_1 \rightarrow$ bus $\rightarrow T_3$ is given by $\Delta = 13$ time unit
- 75% of the bandwidth for S_4 goes unutilized, 50% of bandwidth for S_5 goes unutilized
- The time-triggered nature ensures high time-determinism, but the time predictability comes at the cost of poor resource utilization

TMDA-based time-triggered architecture

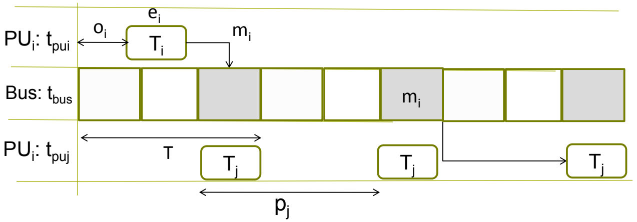
- The communication slots can be categorized as follows:
  - Unassigned slot: these slots are not assigned to any message, but they can be used for future messages.
  - Unused slot: these slots are assigned to a message, but the message is not ready for transmission.
  - Assigned and used slot: a message assigned to such a slot will be transmitted in the assigned slot and will have predictable timing properties.
- The time-triggered architecture is time-deterministic.
- The resource usage is poor in time-triggered design. Moreover, it is inflexible in many cases.
- Poor resource usage and inherent inflexibility make a time-triggered architecture expensive in terms of implementation cost.
- In many real-life scenarios, because of the bandwidth limitation, the time-triggered resources are reserved for safety-critical applications.

Time-triggered synchronous architecture



- When the processors are synchronized with the communication bus, all have the same notion of time, i.e., $t = t_{pu_i} = t_{bus} = t_{pu_j}$
- By appropriate choice of message and task schedules, it is possible to achieve an end-to-end delay $\Delta = e_i + S + e_j$
- Thus, with synchronous time-triggered architecture, it is possible to achieve KNOWN and CONSTANT end-to-end-delay Δ
- E.g., FlexRay based systems

Time-triggered asynchronous architecture

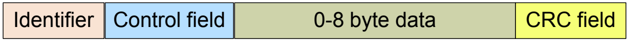


- When the processors are asynchronous to the communication bus, they may not have the same notion of time, i.e., $t_{pu_i} \neq t_{bus} \neq t_{pu_j}$
- The best case is when the end-to-end delay $\Delta_{bc} = e_i + S + e_j$ (same as synchronous case)
- The actual end-to-end delay is constant and $\Delta_{bc} \leq \Delta \leq \Delta_{wc}$
- Thus, with fully asynchronous time-triggered architecture, it is possible to achieve UNKNOWN, BOUNDED and CONSTANT end-to-end-delay

Event-triggered Communication

Event-triggered communication: CAN

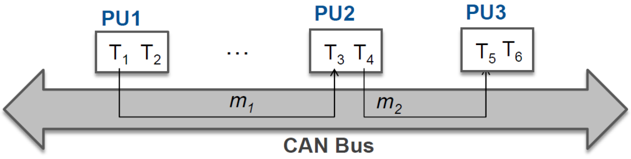
- Controller Area Network (CAN) simple and robust broadcast bus capable of operating at a speed of up to 1 Mbit/s
- Each CAN data frame has the following structure (simplified)



- The identifier of CAN frame has two significance:
  - Identifier indicates the priority of the message
  - It helps the receiving PU to filter out the messages that they not interested in
- The arbitration mechanism employed by CAN means that messages are sent as if all the PUs on the network shared a single global priority based queue. In effect, the messages are sent on the bus according to fixed priority non-preemptive scheduling.
- At any given point in time the message with the highest priority gets transmitted.

Scheduling model for CAN

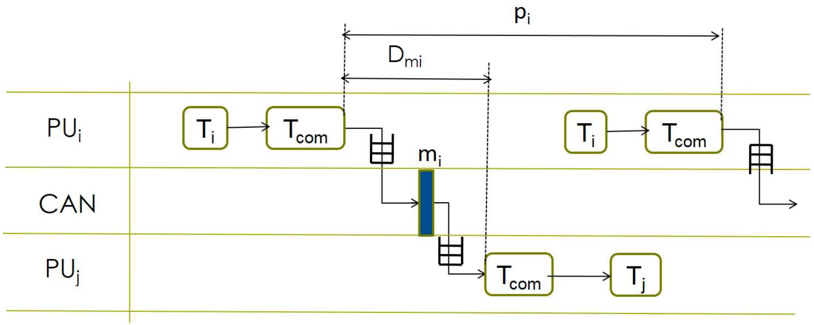
- The system is assumed to comprise a number of PUs connected via CAN. Each PU is assumed to be capable of ensuring that at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration.



- The system is assumed to contain a static set of messages m_i. These messages are generated by one task running on a PU.
- Each message m_i has a fixed identifier and hence a unique priority. As priority uniquely identifies each message

- Each message m_i has total message size s_i bits and has a maximum transmission time over CAN $c_i = s_i \tau_{bit}$ where τ_{bit} is the transmission time of one bit. In general, $10^{-3} sec >> \tau_{bit} > 0$
- A messages m_i is generated by the sending task periodically with period p_i and placed in the global priority based queue with unique priority pri_i. Therefore, the period of the sending tasks is referred to as message period p_i
- Each message has a hard deadline D_{mi}, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving PU that requires it.

Communication over CAN



Schedulability

- The worst-case response time R_{mi}, of a message is defined as the longest time from the initiating event occurring to the message being received by the PU that requires it.
- A message is said to be schedulable if and only if $R_{mi} \leq D_{mi}$ The system is schedulable if and only if all of the messages in the system are schedulable.
- The timing behavior of the CAN messages is considered to be a same as scheduling periodic non-preemptive tasks on an uni-processor.
- The worst-case scenario for CAN message is the one arising at the critical instant when all the messages are generated simultaneously. That is, if the messages are schedulable at the critical instant, then they will also be schedulable in all other scenarios.

Response time analysis for fixed-priority non-preemptive task-sets with application to CAN messages

Response time analysis for CAN messages

- The response time of a message m_i is given by

$$R_{mi} = w_{mi} + c_i$$

  where, w_{mi} = queuing delay, i.e., the longest time before a message m_i gets access to the bus

- The queuing delay w_{mi} is composed of blocking time B_{mi} due to lower priority messages which may be in the process of being transmitted when message m_i is queued:

$$B_{mi} = \max_{k \in lp(m_i)} (c_k)$$

  where $lp(m_i)$ is the set of messages with lower priority than m_i

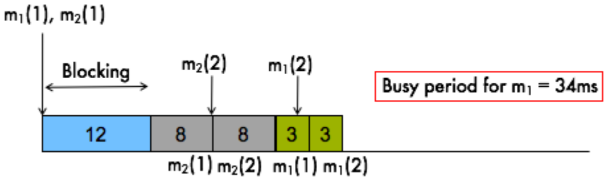
message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1(highest)
m_3	40	30	12	3

- The blocking times are
  $B_{m1} = 12\text{ms}$
  $B_{m2} = 12\text{ms}$
  $B_{m3} = 0$

Busy period

- The busy period t_{mi} of a message m_i is the time duration starting from the critical instant till all the released instances of the message m_i are successfully transmitted.

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1(highest)
m_3	40	30	12	3



Busy period computation

- The busy period can be computed for a message m_i with priority pri_i:

$$t_{mi}^{k+1} = B_{mi} + \sum_{\forall m_j \in hp(m_i) \cup m_i} \lceil \frac{t_{mi}^{k}}{p_{mj}} \rceil c_{mj}$$

  where $hp(m_i) \cup m_i$ is the set of messages with priority of m_i or higher.]

- Initialization: $t_{mi}^{0} = c_{mi}$
- Termination condition: $t_{mi}^{k+1} = t_{mi}^{k}$
- Convergence is guaranteed as long as,

$$U_{mi} = \sum_{\forall m_j \in hp(m_i) \cup m_i} \frac{c_{mj}}{p_{mj}} < 1$$

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1(highest)
m_3	40	30	12	3

Busy period for m_1 (in ms):

$$B_{m1} = 12$$
$$t^0_{m1} = c_1 = 3$$
$$t^1_{m1} = 12 + \lceil \tfrac{3}{30} \rceil 3 + \lceil \tfrac{3}{20} \rceil 8 = 23$$
$$t^2_{m1} = 12 + \lceil \tfrac{23}{30} \rceil 3 + \lceil \tfrac{23}{20} \rceil 8 = 31$$
$$t^3_{m1} = 12 + \lceil \tfrac{31}{30} \rceil 3 + \lceil \tfrac{31}{20} \rceil 8 = 34$$
$$t^4_{m1} = 12 + \lceil \tfrac{34}{30} \rceil 3 + \lceil \tfrac{34}{20} \rceil 8 = 34$$

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1(highest)
m_3	40	30	12	3

Busy period for m_2 (in ms):

$$
\begin{aligned}
&B_{m2} = 12\\
&t_{m2}^{0} = c_2 = 8\\
&t_{m2}^{1} = 12 + \lceil \tfrac{8}{20} \rceil 8 = 20\\
&t_{m2}^{2} = 12 + \lceil \tfrac{20}{20} \rceil 8 = 20
\end{aligned}
$$

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1(highest)
m_3	40	30	12	3

Busy period for m_3 (in ms):

$$B_{m3} = 0$$
$$t_{m3}^0 = c_3 = 12$$
$$t_{m3}^1 = \lceil \frac{12}{30} \rceil 3 + \lceil \frac{12}{20} \rceil 8 + \lceil \frac{12}{40} \rceil 12 = 23$$
$$t_{m3}^2 = \lceil \frac{23}{30} \rceil 3 + \lceil \frac{23}{20} \rceil 8 + \lceil \frac{23}{40} \rceil 12 = 31$$
$$t_{m3}^3 = \lceil \frac{31}{30} \rceil 3 + \lceil \frac{31}{20} \rceil 8 + \lceil \frac{31}{40} \rceil 12 = 34$$
$$t_{m3}^4 = \lceil \frac{34}{30} \rceil 3 + \lceil \frac{34}{20} \rceil 8 + \lceil \frac{34}{40} \rceil 12 = 34$$

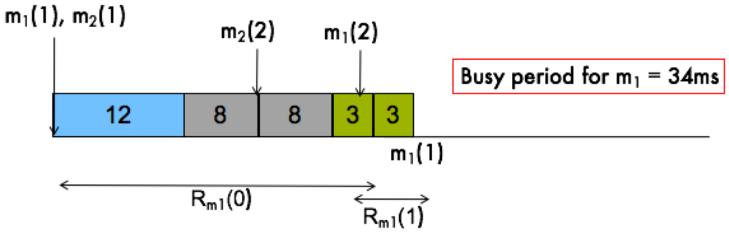
Busy period and response time

- If busy period $t_{mi} \leq p_i$, i.e, one instance the message m_i arrives within busy period, the response time is therefore the same as the busy period.
- When $t_{mi} > p_i$, multiple instance of the message m_i arrive within the busy period. The number of such messages arriving within busy period is given by,

$$Q_{mi} = \lceil \frac{t_{mi}}{p_{mi}} \rceil$$

- In this case, the response time analysis should compute the response time of all Q_{mi} instances. $R_{mi}(q)$ for $q = 0, ...Q_{mi} - 1$
- and the worst-case response time of the message m_i is the longest among them.

$$R_{mi} = \max_{q=0....Q_{mi}-1} R_{mi}(q)$$



- Clearly, there are two m_1 within the busy period 34ms, i.e

$$Q_{m1} = \lceil \frac{t_{m1}}{p_{m1}} \rceil = \lceil \frac{34}{30} \rceil = 2$$

- $q = 0, 1$ and $R_{m1}(0) = 31$ms and $R_{m1}(1) = 4$ms
- $R_{m1} = max(R_{m1}(0), R_{m1}(1)) = 31$ms

How to compute $R_{mi}(q)$?

Computation of $R_{mi}(q)$

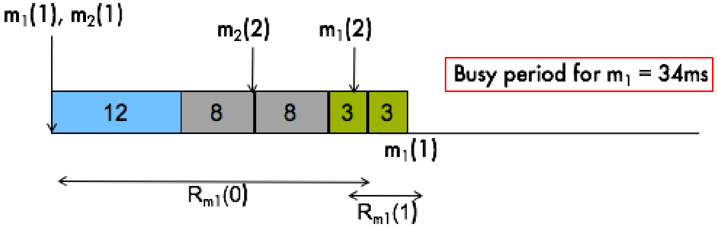
- The longest time from the start of the busy period to instance q beginning successful transmission is given by:

$$w_{mi}^{k+1}(q) = B_{mi} + qc_{mi} + \sum_{\forall m \in hp(m_i)} \lceil \frac{w_{mi}^{k}(q) + \tau_{bit}}{p_m} \rceil c_m$$

$$R_{mi}(q) = w_{mi}(q) - qp_{mi} + c_{mi}$$

- Initialization and termination:

$$w_{mi}^{0}(q) = B_{mi} + qc_{mi}$$
$$w_{mi}^{k+1}(q) = w_{mi}^{k}(q)$$



$$q = 0, B_{m1} = 12, 1 >> \tau_{bit} > 0$$

$$w_{m1}^{0}(0) = 12$$

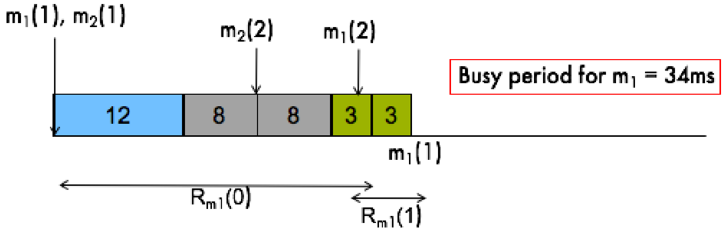
$$w_{m1}^{1}(0) = 12 + \lceil \frac{w_{m1}^{0}(0)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + \lceil \frac{12+\tau_{bit}}{20} \rceil 8 = 20$$

$$w_{m1}^{2}(0) = 12 + \lceil \frac{w_{m1}^{1}(0)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + \lceil \frac{20+\tau_{bit}}{20} \rceil 8 = 28$$

$$w_{m1}^{3}(0) = 12 + \lceil \frac{w_{m1}^{2}(0)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + \lceil \frac{28+\tau_{bit}}{20} \rceil 8 = 28$$

- The response time is given by

$$\begin{aligned} R_{m1}(0) &= w_{m1}(0) - qp_{m1} + c_{m1} \\ &= 28 - 0 + 3 \\ &= 31 \end{aligned}$$



$$q = 1, B_{m1} = 12, 1 >> \tau_{bit} > 0$$

$$w_{m1}^{0}(1) = B_{m1} + qc_{m1} = 12 + 3 = 15$$

$$w_{m1}^{1}(1) = 12 + 3 + \lceil \frac{w_{m1}^{0}(1)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + 3 + \lceil \frac{15+\tau_{bit}}{20} \rceil 8 = 23$$

$$w_{m1}^{2}(1) = 12 + 3 + \lceil \frac{w_{m1}^{1}(1)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + 3 + \lceil \frac{23+\tau_{bit}}{20} \rceil 8 = 31$$

$$w_{m1}^{3}(1) = 12 + 3 + \lceil \frac{w_{m1}^{2}(1)+\tau_{bit}}{p_{m2}} \rceil c_{m2} = 12 + 3 + \lceil \frac{31+\tau_{bit}}{20} \rceil 8 = 31$$

- The response time is given by

$$\begin{aligned} R_{m1}(1) &= w_{m1}(1) - qp_{m1} + c_{m1} \\ &= 31 - 30 + 3 \\ &= 4 \end{aligned}$$

- The response time of m_1 is given by,

$$R_{m1} = \max_{q=0,1} R_{m1}(0), R_{m1}(1) = R_{m1}(0) = 34ms$$

Summary – CAN response time analysis

Blocking time computation

$$B_{mi} = \max_{k \in lp(mi)} (c_i)$$

$$\Downarrow$$

Busy period computation

$$t_{mi}^{k+1} = B_{mi} + \sum_{\forall m \in hp(m_i) \cup m_i} \left\lceil \frac{t_{mi}^k}{p_m} \right\rceil c_{mi}$$

$$\Downarrow$$

Compute number of instances

$$Q_{mi} = \left\lceil \frac{t_{mi}}{p_{mi}} \right\rceil$$

$$\Downarrow$$

Response time of each instance

$$w_{mi}^{k+1}(q) = B_{mi} + qc_{mi} + \sum_{\forall m \in hp(m_i)} \left\lceil \frac{w_{mi}^k(q) + \tau_{bit}}{p_m} \right\rceil c_m$$

$$R_{mi}(q) = w_{mi}(q) - qp_{mi} + c_{mi}$$

$$\Downarrow$$

Response time

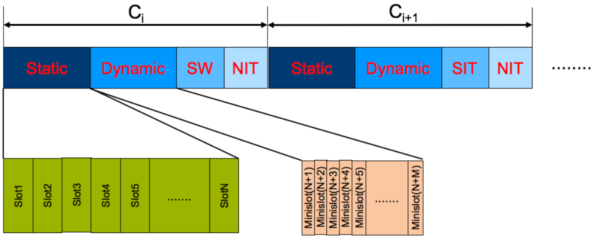
$$R_{mi} = \max_{q=0 \quad Q} R_{mi}(q)$$iscas

Time-triggered and Event-triggered Hybrid Communication

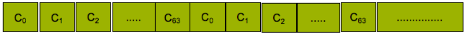
FlexRay - Overview

- Supports both time-triggered and event-triggered communication
- TDMA and Flexible TDMA
- Dual channel for additional bandwidth or redundant links
- 10 Mbits/s transmission rate in each channel
- Data are packed into frames and sent over the bus
- A Flexray frame consists of header (5 bytes), payload (max. 254 bytes) and trailer (3 bytes)
- Distributed clock synchronization mechanism

FlexRay – Communication Cycle



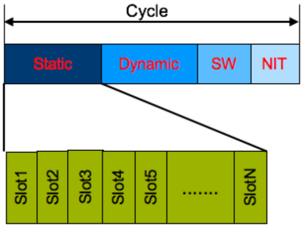
- FlexRay time is organized as an infinite repetition of series of 64 consecutive cycles (0 ... 63)



- Each cycle is of fixed length T_{bus} and consists of:
  - Static segment
  - Dynamic segment
  - Symbol Window
  - Network Idle Time

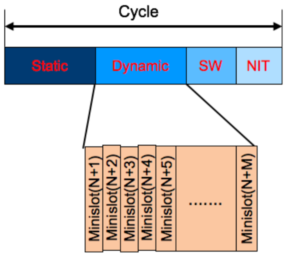
FlexRay – Static Segment

- Time-triggered
- Number of static time slots (N)
- Each slot has a fixed length (Δ) and an assigned ID
- Each data frame assigned to a static slot must be transmitted within the time slot



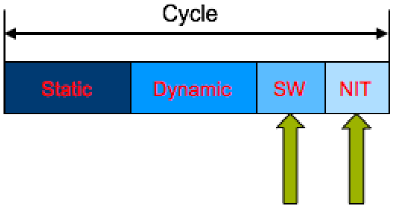
- Hard real-time guarantee possible suitable for control application
- No delays or collision possible
- A slot can be assigned to only one sender node according to FlexRay 2.1 protocol
- Slot multiplexing is possible according to FlexRay 3.0 protocol

FlexRay – Dynamic Segment



- Event-triggered
- Number of small length time slots, i.e., minislots (M)
- Each minislot is of fixed length δ
- A frame may consume one or more minislots
- Slot ID assigned to a minislot is dynamic
- When no frame is transmitted in a minislot, Slot ID is incremented by one after the conclusion of the minislot
- When a frame is being transmitted, Slot ID is incremented only after the conclusion of the last minislot that the frame occupied
- No collision possible.
- No hard real-time guarantee possible

FlexRay–Symbol Window and Network Idle Time



Symbol Window

- FlexRay sends internal control information
- For example, starting the network
- Not a compulsory assignment

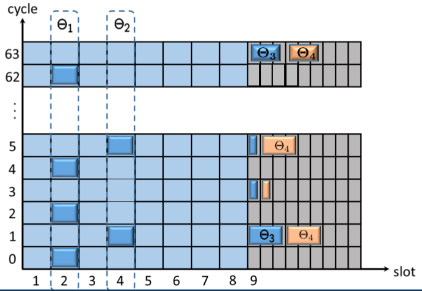
Network Idle Time

- Communication controller executes clock synchronization algorithm during NIT

FlexRay – Frame

- A frame Θ_i is represent by a tuple $\{S_i, B_i, R_i\}$
- S_i represents the slot Id of the slot(s) in which the frame must be transmitted
- B_i represents the base cycle, i.e. the cycle number (0 ... 63) where the frame is transmitted for the first time
- R_i represents the repetition rate, i.e. the number of cycles elapsed between two consecutive transmission of the frame.

$$\Theta_1 = \{2, 0, 2\}$$
$$\Theta_2 = \{4, 1, 4\}$$
$$\Theta_3 = \{9, 1, 2\}$$
$$\Theta_4 = \{10, 1, 2\}$$



FlexRay – Frame

Static segment

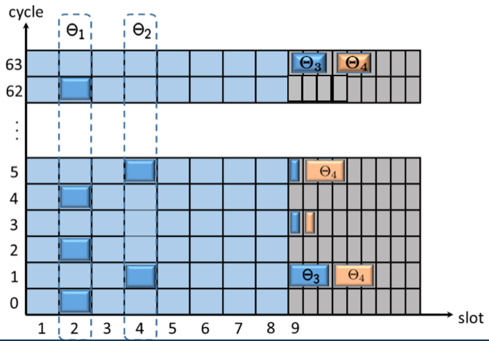
- Frame transmission start time for the k^{th} instant is represented as

$$t(\theta_i, k) = B_i T_{bus} + kR_i T_{bus} + (S_i - 1)\Delta$$

- Frame transmission finish time for the k^{th} instant is represented as

$$\tilde{t}(\theta_i, k) = B_i T_{bus} + kR_i T_{bus} + S_i \Delta$$

$$\theta_1 = \{2, 0, 2\}$$
$$\theta_2 = \{4, 1, 4\}$$



FlexRay – Frame

Dynamic segment

- Earliest frame transmission start time for the k^{th} instant is

$$t(\theta_i, k) = B_i T_{bus} + kR_i T_{bus} + N\Delta + (S_i - N - 1)\delta$$

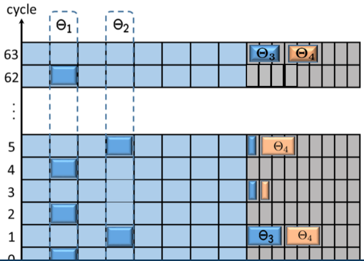
- Latest frame transmission finish time for the k^{th} instant is

$$\tilde{t}(\theta_i, k) = B_i T_{bus} + kR_i T_{bus} + N\Delta + (\sum_{j \in hp_i} (c_j - 1) + S_i - N - 1 + c_i)\delta$$

where c_j is the size of the j^{th} frame in no. of minislots

$\theta_1 = \{9, 1, 2\}$

$\theta_2 = \{10, 1, 2\}$


cycle
Θ1
Θ2
63
62
⋮
5
4
3
2
1
0
Θ3
Θ4
Θ4
Θ3
Θ4