Migration Framework from CAN to FlexRay

Automotive Control Group, Waterford Institute of Technology, Cork Road, Waterford, Ireland
Richard Murphy, Frank Walsh and Brendan Jackman
(rmmurphy@wit.ie) (fwwalsh@wit.ie) (bjackman@wit.ie)

***Abstract*—As FlexRay is implemented in production vehicles (e.g. BMW X5 and 7 series) there is a growing interest within the automotive industry in optimising its utilisation. FlexRay is expected to become the standard network for backbone communications, replacing CAN in this area. However the complexity and cost associated with migrating from existing CAN based systems and designs to FlexRay can prove to be a barrier in its widespread adoption. One of the biggest problems in optimising a FlexRay cycle is formalising the static segment and dynamic segment parameters.**

This paper describes a migration framework for the complete migration from an existing CAN based application to FlexRay based network. This migration framework defines the static (ST) segment size by using basic CAN parameters and performing task graph analysis. The resulting payload is defined before a final ST frame size is obtained. The dynamic (DYN) segment size is verified by determining the worst case response time of tasks operating in this segment. A sample adaptive cruise control (ACC) application is implemented to verify the framework.

I. INTRODUCTION

Modern consumers are seeking improved safety features and increased infotainment when purchasing an automobile. The increased use and sophistication of distributed electronic control systems in the automotive industry has resulted in rising traffic volumes on in-vehicle networks. Implementing these features among existing applications on established predominant automotive protocols (e.g. CAN), will prove a challenge. Due to CANs ET (Event-Triggered) nature, as the bus load approaches capacity all tasks with lower priority will find it difficult to access the bus [1] to complete operation.

This growing communication demand stimulated the establishment of the FlexRay consortium in 2000 and the development of the FlexRay protocol[2]. The FlexRay communications protocol aims to address the demands of such future applications by providing the following features;

- Synchronous and asynchronous data transmission
- Support of a fault tolerant scalable time-base
- Scalable electrical/electronic architectures supporting a multiple of platforms
- Single channel gross data rate of 10Mbits/s
- Arbitration free transmission
- Support for bus and star topologies
- Fast error detection and signalling
- Support of wake-up and sleep functionality via the bus
- Deterministic data transmission with guaranteed message latency and message jitter
- Support for redundant transmission channels

FlexRay provides higher data transfer rates, determinism and fault tolerance not available directly using the CAN protocol. FlexRay is configurable in numerous network topologies such as point-to-point, passive star, linear passive bus, active star network, cascaded active stars and hybrid topologies. However these features come at an increased cost when compared to CAN as FlexRay is still a relatively new protocol so initial purchasing and development costs are still high. Even though FlexRay has many features not available on CAN it is not envisaged that FlexRay will completely replace CAN[3] as illustrated in Figure 1. Both CAN and FlexRay can be implemented side by side through the use of gateways [4] or complete migrations.

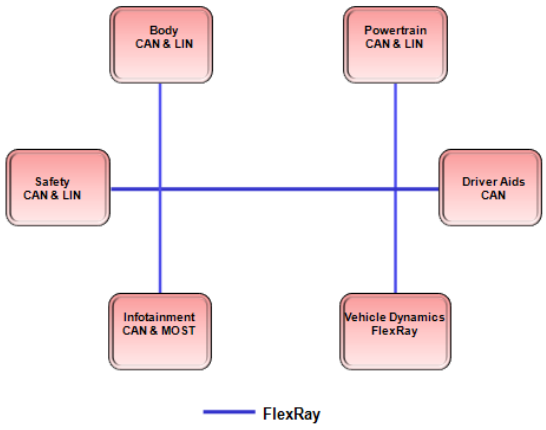

Figure 1: Possible FlexRay Usage

The paper is organised as follows. Section II covers related works from other authors. This includes work into FlexRay frame parameter definition. Section III contains the actual migration process. Section IV contains the case study. Section V contains the results and the paper is concluded with section VI the conclusion.

II. RELATED WORKS

In [3] the author takes the approach from the view point of configuring the minimum number of ST slots (2 synchronisation slots) and the rest is implemented in the DYN segment. This approach works but the ST segment of the FlexRay cycle is completely unutilised. In [5] the author examines the DYN segments performance explicitly. This is done using a markov chain based evaluation. Because only the

Sponsor: Sumitomo Electrical Wiring Systems - Europe (SEWS-E), YSTRAD

DYN segment is examined the evaluation technique does not comprehensively cover the complete FlexRay cycle. Evaluation platforms are available from companies such as Fujitsu [6]. Other authors have approached scheduling from the view point of just utilising the ST segment such as [7] and [8]. In [7] the author uses a genetic algorithm (GA) approach to scheduling. First the algorithm is generated then refined by means of optimisation, crossover and mutation. The GA is verified and found to improve on past approaches. The approach in [8] uses deadline analysis to synthesise task times. By clustering messages and slot reuse the author demonstrates improved utilisation of ST segment bandwidth under tested conditions.

While the previously mentioned works only deal with certain aspects of the FlexRay cycle [9] gives a more comprehensive approach using holistic analysis techniques. Again as with previous examples validation is carried out through simulation.

In [10] the author provides some of the FlexRay cycle parameters used in the electronic damper control in the BMW X5. The Goal when defining the FlexRay system parameters was to have a *"Constant parameter set for all series projects to support carry over of ECUs"*. Using a 10Mbit/s baud rate and a FlexRay frame cycle time of 5*ms,* comprising of a ST segment size of 3*ms* and a DYN segments size of 2*ms*. With possible repetition cycles for frames in the ST segment being 2.5*ms,* 5*ms*, 10*ms*, 20*ms*, 40*ms*, this represents a base period of 2.5*ms*. This allows a configuration so that all the other values are multiples of this base period and the DYN segment can be freed for less critical diagnostics messages.

III. CAN - FLEXRAY MIGRATION

This paper adds to works previously undertaken in the area of scheduling FlexRay frames. Both the ST and DYN cycle segments are accounted for in this framework, while other works focus on utilisation of certain aspects of the FlexRay cycle structure as previously stated in section 2. The approach taken here uses task graph analysis in determining static slot sizes and hence the static segment size. A response time analysis technique is used in determining if dynamic segment size is appropriate for it requirements. While guaranteeing successful transmission the proposed methods result in a high degree of redundancy in the system. Where this work differs from other works described above is that the CAN and FlexRay parameters are tested on hardware and not simulated. This has the advantage of uncovering discrepancies that would not been apparent through simulation.

A. CAN and FlexRay Comparison

Table 1 gives a brief overview of the basic features of CAN and FlexRay. FlexRay has the advantage over CAN in areas of protocol type due to it containing ET and TT (Time-Triggered) properties, data rate due to FlexRays 10Mbit/s on two channels (redundancy). FlexRay features complete fault tolerance as opposed to only having fault tolerance on low speed CAN. Due to CAN being a mature protocol and having fewer complexes than FlexRay it is more attractive for the designer to use.

TABLE 1
CAN – FLEXRAY COMPARISON

Feature	CAN	FlexRay
Protocol Type	Event-Triggered	Time and Event Triggered Segments
Channels	1	2
Data Rate	1MBit/s max	10Mbit/s max on 2CH
Costs	Low	High
Complexity	Not Overly Complex	Complex Protocol
Fault tolerance	Yes (Low Speed CAN)	Yes
Network Management	By Software	By Hardware through Bus driver or Bus guardian

B. FlexRay Communication Protocol

FlexRay while offering improved data throughput, determinism, redundancy and fault tolerance; this comes at a cost of complexity as well as previously mentioned increased monetary cost. The higher monetary cost is a feature with all new products. This will reduce as FlexRay matures and is implemented on a wider scale. This increased complexity [11] is derived from a FlexRay frame containing both a static (ST) and a dynamic (DYN) segment amongst other features. The ST segment is based on time-triggered TDMA type protocol whereas the DYN segment is based on an event-triggered flexible TDMA (FTDMA) type protocol. Each FlexRay cycle is concluded by a communications free period made up of the symbol window and/or Network Idle Time (NIT) as illustrated as just NIT in Figure 2. In the ST segment all slots are the same size. A ST frame can transmit if the ST frame ID matches the ST slot ID. In the DYN segment a minislot size is defined at compile time also. A DYN frame transmits if the DYN frame ID matches the DYN slot ID. If a DYN message does not use its slot a period of 1 minislot is used so as to allow the minislot counter to increment.

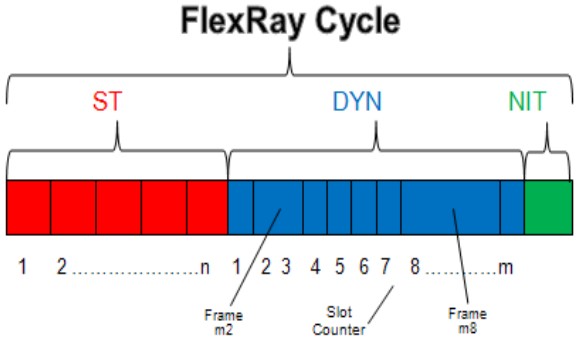

Figure 2: FlexRay Cycle Structure

A message can occupy more than one minislot as illustrated in

Figure 2 where message 2 occupies minislot 2 and 3. A message will not transmit in the DYN segment if the minislot counter value is greater than the *pLatestTx value*. The NIT can be used for synchronisation purposes or to transmit a wake up symbol.

C. Migration Requirements

As already stated, the migration procedure is designed to transition pre-existing CAN based systems to FlexRay. Because CAN systems physical architecture invariably consists of a bus topology, it is assumed that this topology is maintained as part of the migration procedure. Each CAN application is logically abstracted as a task graph in order to analyse and extract input and output parameters for the migration procedure. A simple example of a sensor/processor/actuator task graph is shown in figure 3 where each node represents a CAN task (T_i) and each edge represents a directed communication link between nodes. The arrow indicates the direction of data transfer. Here we have the task graph release time r_i the deadline time D_i. The task graph starts at task T_i and ends at task $T_{i+n.}$

Deciding on which tasks are assigned to the ST or DYN segments is done by individually assigning application tasks into critical and non-critical priorities. In this framework all critical tasks (e.g. Brake-by-wire) are mapped into the ST segment and all non-critical (e.g. air conditioning) tasks are mapped into the DYN segment.

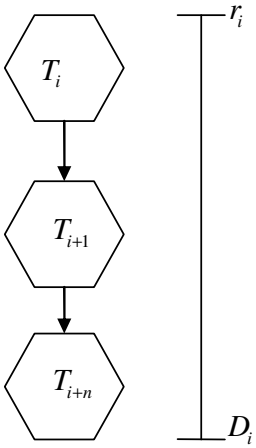

Fig 3: Basic Task Graph

Physically each task is allocated to a specific processor with inter-processor messages requiring transmission across the underlying communication network. Each task in the CAN application can have the following time based properties;

- Task (T_i)
- WCET (w_i)
- Task Deadline Times D_i and Release Times r_i
- Task Period (Task Frequency)

The task worst case execution time (WCET) forms a central part in this framework. This is done so as many delays as possible are taken into consideration to obtain improved validity of the results.
The initial input parameters of the migration framework are provided from the existing CAN application.

D. Parameter Calculation

A key feature of this framework is moving from task analysis to message analysis by calculating the properties of all inter-processor messages. This is required because the migration process decouples messages from tasks through task graph analysis. Each tasks execution time is calculated from when the task is signalled to execute until it has completed execution. To initially schedule a task parameters required from the task graph are r_i and D_i.

This is necessary because, as illustrated in Figure 3, execution of task T_i can potentially delay task T_{i+1} from executing. This results in equation 1.

To schedule an intermediate task;

- An intermediate task deadline is represented by d_i
- The intermediate tasks execution times are required prior to intermediate task scheduling
- The release time of the first intermediate task (T_{i+1} in Figure 3) is derived from the deadline of the previous (initial) task (T_i)
- This intermediate task deadline is then obtained by adding its release time to its execution time
- This process is repeated for each task resulting in an initial release time and deadline time for each intermediate task.
- The total amount of slack for re-allocation is illustrated in equation 1 where c_i is the execution time of a task T_i along the chosen path.

$$TotalSlack_i = D_i - \sum c_i \qquad \text{Eqn (1)}$$

To determine the final task parameters, any slack in the system is re-allocated equally among each task on a particular path x of the task graph as illustrates in equation 2.

$$slack_i = \frac{TotalSlack_i}{x} \qquad \text{Eqn (2)}$$

When a task is assigned a new release time or deadline time, the task graph is updated to include these new values. The updated times are removed from the original task graph before the next task graph path is analysed. The path resulting in the longest r_i and d_i times is chosen to propagate through the system. This is because all other value will return quicker paths and times.
This process is demonstrated using the example task graph illustrated in Figure 4.

The execution time for each task is shown in Table 2.

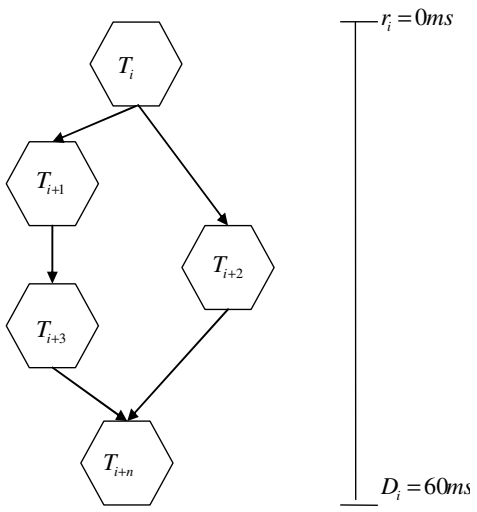

Figure 4: Sample Task Graph

TABLE 2
TASK GRAPH EXECUTION TIMES

Task Number	*Execution Time*
T_i	10
T_{1+1}	5
T_{i+2}	8
T_{i+3}	6
T_{i+n}	8

The release and deadline times of each task are then updated after being recalculated.
Path T_{i+2} is then calculated separately with its release time determined by the deadline time of the previous task T_i. This gives a release time of 17.25 and deadline time of 45.75. The final task release and deadline times are shown in Table 3.

TABLE 3
FINAL TASK TIMES

Task Number	Release Time	Deadline Time
T_i	0	17.25
T_{1+1}	17.25	24.2
T_{i+2}	17.25	45.75
T_{i+3}	30.5	45.75
T_{i+n}	45.75	60.0

In the case of a multi-rate system the cycle value of the least common multiple (LCM) of all coupled applications is required to guarantee the timely execution of all tasks while maintaining message periods. An example of this is if there are two task graphs with periods of 2ms and 5ms respectively. A hyper-cycle of 10*ms* is required to guarantee transmission of all messages.

E. Message Analysis

Message analysis can only be carried out once the initial task parameters have been determined. Message analysis prepares for message discretisation. Equation 3 can be used as an initial check to see if the individual task parameters are valid.

$$w_i \leq d_i - r_i \qquad \text{Eqn (3)}$$

A key factor in determining timing properties of a message is the maximum amount of time available to transmit that message once the source task has completed execution. If a task T_i is a message source then the task must complete execution and transmit the resulting message m_i before the task deadline, d_i expires. If the message delay is greater than the deadline time for that message is not feasible to transmit that message. Therefore once the deadline expires an allocated transmission "slot" will not be available until the next communication cycle. Each messages deadline $td(m_i)$ is determined by subtracting the task release time and WCET [8] from the task deadline as illustrated in equation 4. Where $td(m_i)$ is the transmission deadline of message *i*.

$$td(m_i) = d_i - r_i - w_i \qquad \text{Eqn (4)}$$

The primary factors affecting message transmission are;

- If a node attempts to transmit
- Available bandwidth
- Message size

Bus contention is not required for consideration due to the deterministic nature of message transfer. The transmission delay may be calculated using equation 5. The $size(m_i)$ and Bus_{speed} are in units of bits.

$$transmission\ delay = \frac{size(m_i)}{Bus_{speed}} \qquad \text{Eqn (5)}$$

In the ST segment, task 1 (T_1) transmits message 1 (m_1) so message 1 is assigned to ST slot 1 and message 2 (m_2) is assigned to ST slot 2 up to message *n* being assigned to ST slot *n*.

F. Payload Optimisation

The FlexRay frame is composed of the Header, Payload and Trailer segment as per the FlexRay specifications [12]. The header and trailer are considered overhead because the data is used for transmission but not used by the application to carry out any function. Figure 5 illustrates an example of the overhead required in relation to the payload size.

The overhead associated with this was calculated at 14bytes. The 14bytes overhead frame was composed of:

- 5 bytes for the header
- 3 bytes for the CRC (Cyclic Redundancy Check)
- 2 byte max of a TSS (Transmission Start Sequence)
- 4 bytes for the clock and security (there is a minimum variance required between messages from different nodes so there is no overlap. Includes safety margin of $4\mu s$)

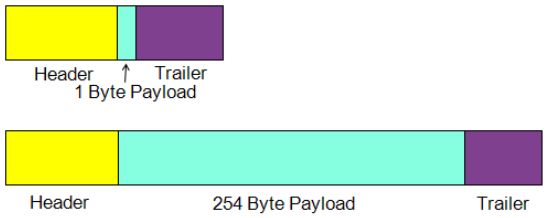

Figure 5: Frame Overhead

Frame payload dictates the size of the static slot. Therefore it is an important optimisation parameter because a payload value larger than what is required can lead to underutilisation of bus bandwidth. An example, when transmitting messages of up to 6 bytes in a static frame that can accommodate up to 10 bytes results in suboptimal use of available bandwidth. Furthermore, choosing a smaller payload size can enable the designer to choose a smaller static slot size resulting in finer granularity to the static segment. The optimal scenario is maximising data transmission while minimising transmission overhead.

Equation 6 is used to determine the number of frames required to transmit a message at the chose payload size. Here *FR frames*$_n$ is the number of frames required at the chosen payload size for the transmission of a complete message cycle $m_i \ldots m_x$. This procedure involves rounding up to the nearest whole integer value.

$$FR\ frames_n = \left\lceil \frac{m_i\ size}{payload\ size} \right\rceil \quad \text{Eqn (6)}$$

The total number of bytes for complete transmission gives a clear indication which combination of, Number of Messages *n*, Payload Size and Frame Size are the most appropriate. The number of Total bytes is given in equation 7.

$$Total\ Bytes = Frame\ Size \times n\ messages \quad \text{Eqn (7)}$$

By graphing the Bytes per Cycle v's Frame Size the general graph profile is as illustrated in Figure 6.

As the frame size increases initially, the number of bytes per cycle drops rapidly (region 1). After this initial period the difference between the number of bytes per cycle in consecutive frames sizes gets smaller (region 2). This is because the same amount of data is sent but fewer frames are required. By transmitting fewer frames, less overhead is incurred. In the final region (region 3) of the graph the number of bytes per cycle starts to increase again. This increase is not as dramatic as the initial decrease in region 1 and is due to an increased payload size leading to an increased frame size, while still transmitting the same amount of data.

The FlexRay frame size determines the slot size. The optimal frame size is not immediately apparent. This is because the optimal frame size is not necessarily the one associated with the minimum number of transmitted bytes. By choosing a large frame size the through put of data is increased. This in turn reduces the granularity of the FlexRay cycle. If the system designer chooses a smaller frame size this means there are higher overheads associated with sending the same amount of data than if a larger frame size was chosen. The final frame size is to be chosen by the designer depending on specific needs and requirements.

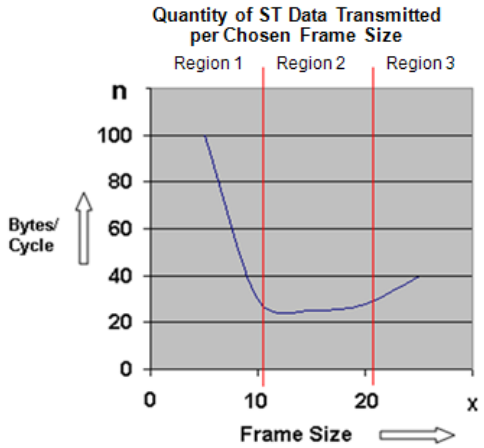

Figure 6: Graph of Bytes per cycle v's Frame Size

G. Slot Size Definition

By obtaining the message size in a discrete format the message size can be represented as a function of discrete slots rather than as a function of time. Each ST slot is the same size (in terms of the number of macroticks (MT)) in a frame according to the FlexRay specifications [12]. The ST slot is composed of an integer multiple of MTs.

The message m_i period *period*(m_i) is equal to its task deadline *td*(m_i). This guarantees enough time for the message to complete communication.

The calculated optimal slot size *gdStaticSlot* is the frame size in bytes (*size* (*payload*$_i$) and *size* (*overhead*$_i$)) divided by the bus speed Bus_{speed} as illustrated in equation 8.

$$gdStaticSlot = \frac{size(payload_i) + size(overhead_i)}{Bus_{speed}} \quad \text{Eqn (8)}$$

Restrictions on the actual obtainable slot size are governed by Constraint #15 in Appendix B of the FlexRay specifications v2.1 rev A. [12].

To discretise the static slot size equation 9 is used, where *gdStaticSlot* is the static slot size.

$$td(M_i) = \left\lceil \frac{td(m_i)}{gdStaticSlot} \right\rceil \quad \text{Eqn (9)}$$

The discretised slots at this point enable the message size to be displayed as a function of the ST segment which is more practical when configuring the ST segment of the FlexRay frame. The discretised slot duration is denoted as $td(M_i)$ to differentiate it from the un-discretised message delay.

To guarantee the message m_i deadline there must be periodicity $period(m_i)$ between successive messages. The maximum distance between successive transmission slots m_i is to be equal to the $period(m_i)$ [8].

A base period p_{base} is then selected. The smallest $td(m_i)$ is chosen as the initial base period p_{base}. From this value message periods in multiple harmonics are chosen which meet the periodicity requirement. The base period value is chosen in the format of transmission slots. A message integer period greater than the maximum number of transmission slot intervals results in a violation of the periodicity constraint. All messages transmitted in the ST segment are guaranteed to meet their deadlines due to the TT nature of the ST segment. With this in mind it is important to configure the FlexRay cycle to give the DYN segment as much opportunity to transmit as possible. If the cycle period can be reduced without affecting ST and DYN transmission times adversely this should be done so. An example of this is if a cycle period of 10*ms* enables deadlines to be met but a period of 5*ms* also results in deadlines being met the 5*ms* cycle period should be chosen. This allows the DYN tasks the opportunity to gain access to the bus twice as often as if a 10*ms* cycle period was chosen.

Once the base period is selected and the discretised delay is chosen the parameters need to be validated. Equation 10 is used to validate that the chosen parameters meet the required deadlines.

$$2^k \cdot p_{base} \le td(M_i) < 2^{k+1} \cdot p_{base} \quad \text{Eqn (10)}$$

The base period value can be modified but all modifications still have to ensure that the periodicity constraint and the distance constraint are met. The procedure is summarised in Figure 7 algorithm.

ST Segment Scheduling Algorithm ()

Initialise initial CAN parameters *WCET, r_i, D_i, Task period*

Perform Task Graph Analysis

Obtain intermediate tasks r_i and d_i values
Re-allocate slack to path undertaking analysis
Obtain r_i and d_i times along chosen task graph path

If (using multirate system)

{

LCM of interacting task graphs is required to guarantee timely transmission

}

Update new r_i and d_i times per path analysed

Determine message delay $td(m_i)$
Find optimised payload and configure frame size

Heuristically chose optimised frame size

Determine slot size and discretised

Adjust message periods ensuring periodicity and distance constraint.

Figure 7: ST Scheduling Algorithm

A. *Dynamic Task Analysis*

In calculating the DYN segment size the first parameter required is the minimum time for the complete FlexRay frame. This is obtained from equation 11. This calculates the time taken to transmit all the data with no delays.

$$Min_{FR}(t) = ST_{bus} + messageID_i \ldots messageID_n + NIT \quad \text{Eqn (11)}$$

The $MIN_{FR}(t)$ value will help tell if there are enough slots for the messages that will require transmission through the DYN segment. Equation 12 checks if there are enough slots in the DYN segments at the current frame configuration to give each DYN message a chance for transmission. Here $FR(t)$ is the size of the FlexRay frame.

$$No\_of\_DYN_m \le FR(t) - (ST_{bus} + NIT) \quad \text{Eqn (12)}$$

To obtain a realistic DYN segment, delays to the DYN messages need to be calculated. Worst case response time analysis R_m is used to determine the length in time of a dynamic messages response. This different (to the ST segment) approach is required because the DYN segment is event-triggered. Some prerequisites include that only one node can transmit on the bus at any one time in a slot (either static or dynamic). The node determines when the slot counter is equal to the value of a frame identifier. By allocating one slot to at most one node this avoids any conflicts that might occur. The minislot counter value has to be less than the *pLatestTx* value which is defined in constraint #36 of appendix B in the FlexRay specifications v2.1 rev A [12]

Each message is assumed to have an overhead as calculated for the static segment in section F (14 bytes). Adding the overhead to the message size gives the frame size for

transmission per message. The dynamic segment is composed of an integral multiple of the minislot length. The size of each minislot in the dynamic segment can be any integer value between 2 MT and 63 MT as defined in the FlexRay specifications v2.1 rev A appendix B [12].

1) Message Cycle Delays

The earliest possible time to transmit a DYN message is after the ST segment has finished.

The worst-case response time *Rm(t)* (equation 13) of a dynamic message is calculated from [9] the delay during one bus cycle if its slot has passed. The parameters;

- δ_m the worst-case delay caused by the transmission of static messages and higher priority frames
- w_m is the delay caused by static messages and higher priority dynamic frames.
- C_m the communication time

$$R_m(t) = \sigma_m + w_m(t) + C_m \qquad \text{Eqn (13)}$$

The communication time is determined from the message frame size $Fmessage_i$ divided by the bus speed Bus_{speed} as illustrated in equation 14. Here $Fmessage_i$ and Bus_{speed} are in bit form.

$$C_m = \frac{Fmessage_i}{Bus_{speed}} \qquad \text{Eqn (14)}$$

The worst-case scenario of when a message can be generated is if it is generated immediately after the slot with its frame identifier has passed. The worst-case delay δ_m can be written as equation 15. The length of the static segment is ST_{bus}.

$$\sigma_m = Min_{FR} - (ST_{bus} + (messageID_i.gdMinislot) + NIT) \qquad \text{Eqn (15)}$$

Next w_m is defined in equation 16, as blocking by static messages, *hp(m)* higher priority messages and any unused dynamic slots which gives a delay of one minislot *gdminislot ms(m)* each The single minislot is required to enable the minislot counter to increment to the next value. For this calculation the worst case delay occurs if the message requires transmission at the moment the *pLatestTx* value is the same as the minislot counter. Therefore all minislots after this value cannot be used for transmission.

The frame identifier also determines the frames priority in the DYN segment.

$$w_m(t) = ST_{bus} + hp(m) + pLatestTx + NIT \qquad \text{Eqn (16)}$$

The values obtained can be discretised to determine the DYN segment size in slots but is not necessary due to different messages occupying different amounts of minislots.

Also included in the FlexRay frame is the NIT. This value can be calculated using constraint #27 and the symbol window is calculated from constraint #16 in Appendix B [12]

The DYN segment algorithm is illustrated in Figure 8.

DYN Segment RTA Algorithm()
Initialise predefined parameters
{
$MessageID_i$, ST_{bus}, FR NIT, $pLatestTx$
}
Find first possible transmission time after ST segment
Determine delay if message slot has just passed
Determine delay due to *hp(m)* and *ms(m)*
Determine C_m
Combine delays to for total WCRT *Rm(t)*

Figure 8: DYN Segment Algorithm

IV. Case Study

The migration procedure was applied to an advanced automotive control application as detailed in Figure 9 and Table 4. Experimental validation was carried out using the following system specification. Initially performance results are obtained for the CAN implementation under various traffic conditions. Similar results are recorded for the migrated FlexRay based system.

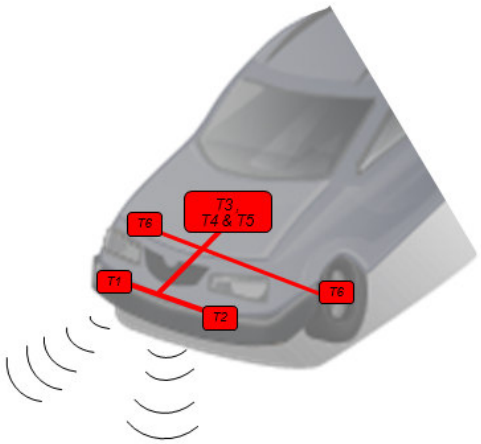

Figure 9: ACC Example Task Configuration

A. System Design

A two node system was tested with each task assigned to a node depending on its function. The tasks dealing with "actions" were placed on node 1 (N1) and the tasks performing computational duties were placed on node 2 (N2). Channel A on N1 was connected with channel A on N2 through an active-passive star configuration. Channel B was set up with a similar configuration where channel B on N1 is connected with channel B on N2. Bus bandwidth of 10MBit/s was chosen for FlexRay and a bandwidth of 125kbit/s was chosen for CAN. Both test configurations are illustrated in Figure 10.

TABLE 4
ACC PROPERTIES

Task Number	Operation
T1	Vehicle Velocity
T2	Distance to Vehicle in Front
T3	Calc Relative speed of Vehicle in Front
T4	Calc Desired Velocity
T5	Calc Absolute Throttle Value
T6	Actuate Throttle and Breaks

B. *Experimental Environment*

Both test configurations (CAN and FlexRay) were set up on two Fujitsu SK-91F467 FlexRay development boards, with each representing one node. The development board contained an MCU (microcontroller) and separate CC (communications controller). This was connected to the FlexRay physical layer via physical layer driver (FlexTiny FT1080) as per the FlexRay specifications.

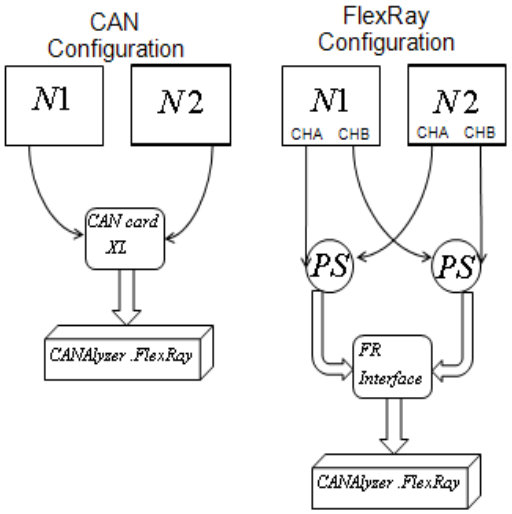

Figure 10: CAN and FlexRay Test Configuration

The tasks *T1* and *T2* are assigned WCET of 0.0*ms* because the start of the application is signalled once one of these values has been received. In reality there is some delay from the time the sensor detects a value until it is passed but this value is considered negligible in this test. Figure 11 illustrates the task graph and associated CAN parameters. The CAN parameters as they were obtained from task graph analysis are shown in Table 5, after the slack has been redistributed.

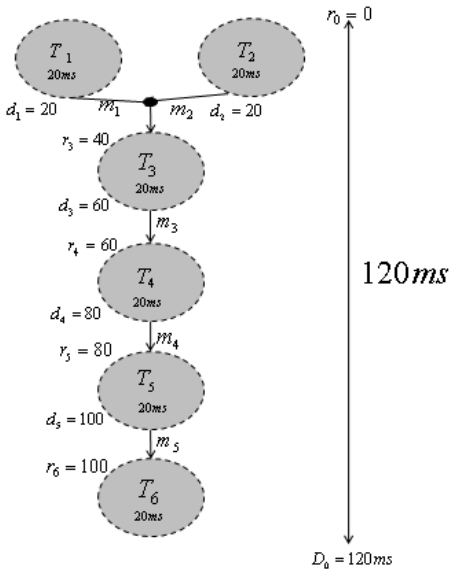

Figure 11: CAN Task Graph

TABLE 5
ACC PARAMETERS

Task #	WCET (ms)	Release Time (ms)	Task Deadline (ms)	Slack per task(ms)	Task Scheduling Deadline (ms)
T_1	0.000	0.000	0.020	0.019	0.020
T_2	0.000	0.020	0.040	0.019	0.040
T_3	0.006	0.040	0.060	0.019	0.054
T_4	0.002	0.060	0.080	0.019	0.078
T_5	0.006	0.080	0.100	0.019	0.094
T_6	0.002	0.100	0.120	0.019	0.118

C. *CAN to FlexRay Migration*

Table 6 shows the message sizes in bytes and the transmission delay for each message. The solution is considered feasible at this stage due to the transmission delay being less than the deadline delay td(m_i). Using the message sizes as specified in Table 6 results in the graph illustrated in Figure 12. A frame size of 24 is chosen which results in a payload of 5 two-word-bytes

The message periods are now discretised. A slot size of 40μs is selected. A slot size of 20 μs is extracted from the framework as per equation 8. The implemented slot size was modified due to the minimum achievable slot being 33μs (Decomsys designer restriction as per FlexRay specifications constraint #14), also a 40μs slot size yields an even slot count on all messages so there is no requirement to round off the number of slots. If the obtained value of 20μs was used this would give a minimum period of 700 slots as opposed to the 350. The discretised $td(m_i)$ is illustrated as the number of slots. This is illustrated in Table 7.

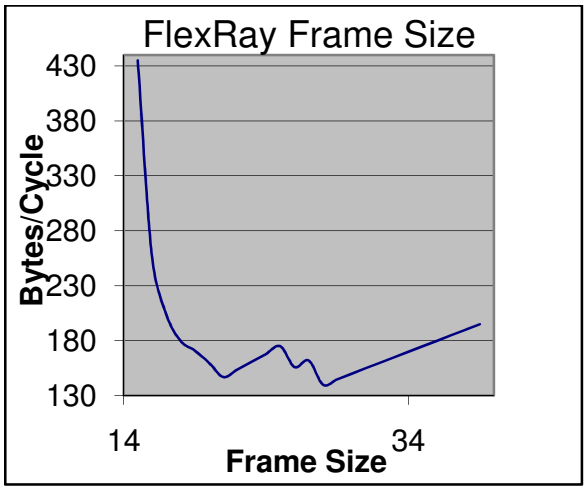

Figure 12: Optimal FlexRay Frame

TABLE 6
MESSAGE ATTRIBUTES

Message #	Max Size (bytes)	Transmission Delay (µs)	Transmission Delay (Slot)
m_1	10	8	1
m_2	10	8	1
m_3	10	8	1
m_4	10	8	1
m_5	10	8	1

TABLE 7
MESSAGE DEADLINES

Message #	DEADLINE $td(m_i)$ (MS)	$td(m_i)$ (SLOTS)
m_1	0.020	500
m_2	0.020	500
m_3	0.014	350
m_4	0.018	450
m_5	0.014	350

With the smallest message period being 14*ms*/350 slots, this is used as the base period. This value satisfies equation 10.

To evaluate the size of the DYN segment two DYN messages were transmitted at random times with constraints. The constraints ensure transmission was in the range of very 2*ms*-20*ms*. A minislot size of 6*µs* was chosen by constraint #14 in the FlexRay specifications.
Table 8 contains the parameters as calculated per equation 13. The values can be discretised and calculated as a function of number of minislots. This is illustrated in Table 9

TABLE 8
DYN MESSAGE ATTRIBUTES

Message #	Message Size (Bytes)	σ_m (ms)	$w_m(t)$ (ms)	C_m (ms)	$R_m(t)$ (ms)
m_7	4	1.4806	0.303	0.0144	1.1980
m_8	4	1.4806	0.317	0.0144	1.8124

TABLE 9
DYN MESSAGE ATTRIBUTES (SLOT)

Message #	Message Size (Bytes)	σ_m (slots)	$w_m(t)$ (slots)	C_m (slots)	$R_m(t)$ (slots)
m_7	4	247	51	3	300
m_8	4	247	53	3	303

V. RESULTS

The results section demonstrates the findings obtained through implementation of the framework as described above. As FlexRay contains CH A and CH B this paper deals with CH A as the primary channel for ST message transfer and CH B as the redundant channel. DYN messages are only assigned to CH B. This is the chosen set up because ST data is considered of a critical priority, while messages transmitted on the DYN segment are not to be considered as critical in this test case. All results are recorded over a 30 second sample period.

In the CAN set up a task graph deadline of D_{CAN} = *120ms* exists. After undergoing task graph analysis the task graph deadline becomes $D_{FlexRay}$ = *84ms*. The value is obtained from modifying the FlexRay message period to 14*ms* from the CAN value of 20*ms* as illustrated by the findings in Figures 13 and 14. Figure 13 contains the CAN results and figure 14 contains the FlexRay results. Figure 13 shows a message maximum execution time of 7.845*ms* compared to deadline time of 20*ms*. This maximum Figure is taken after the longest WCET of 6*ms* is applied to the task. FlexRay messages result in different cycle values depending on the same WCETs as in the CAN test. The maximum message cycle is 7.0380*ms* with a WCET of 6*ms* in the FlexRay test.

Each FlexRay ST message meets its deadline of 14*ms* as shown in Figure 14. The same messages in CAN also meet their deadlines but message times are more consistent in FlexRay
From Figure 14 it is observed that all message deadlines are easily met including where message times fluctuate in CAN. At higher data rates CAN messages would be susceptible to message times increasing where as in FlexRay these messages times are always guaranteed. To get a clearer indication of if the application is as successful on FlexRay as CAN we examine the applications cycle times.

Examining the complete applications cycle times, Figure 15 represents the CAN data and Figure 16 represents the FlexRay data. The CAN cycle has a deadline of 120*ms* but has completed execution by a maximum time of 20.478*ms*. The application implemented in FlexRay has a deadline of 84*ms* but completes execution with a maximum time of 22.947ms. This shows redundancy in the system of 61*ms* in FlexRay and 99*ms* in CAN. Even at this maximum cycle time the deadline of 84*ms* is not close to being exceeded.

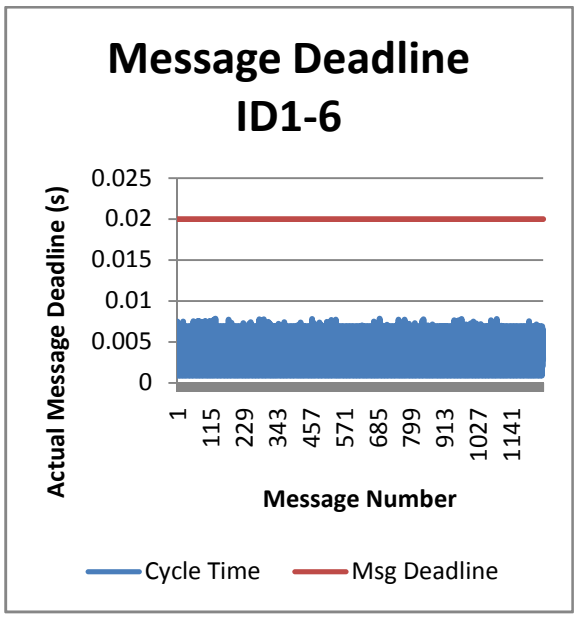

Figure13 : CAN Message Times

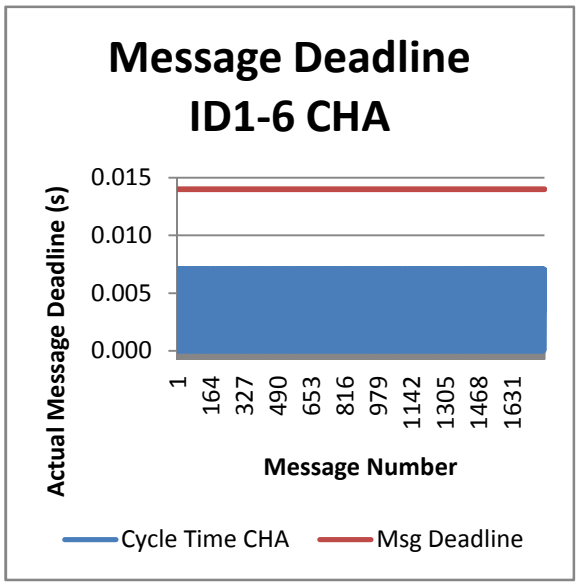

Figure 14: FlexRay Cycle Times

Table 10 gives a detailed breakdown of the FlexRay ST task parameters. As task messages m_1 and m_2 are the initial times and have no precedence constraints there is a minimum delay of zero. Column four shows the actual maximum execution time.

The Framework allows the extraction of FlexRay configuration parameters. The parameters shown in Table 11 are required to configure the FlexRay frame for successful transmission. The setup includes 6 tasks in the static segment and 2 tasks in the DYN segment. A MT was set at 1μs. This minimum configuration results in a FlexRay cycle of 1.750*ms*.This value is obtained from the p_{min} value of 14*ms*.

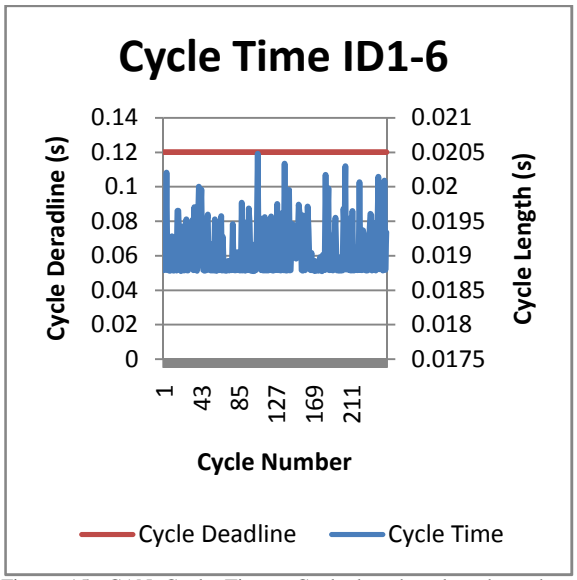

Figure 15: CAN Cycle Times. Cycle length values have been scaled to give a clearer representation.

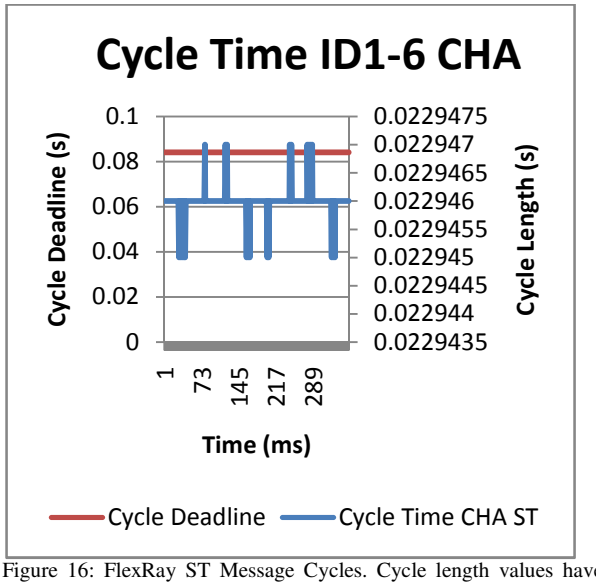

Figure 16: FlexRay ST Message Cycles. Cycle length values have been scaled to give a clearer representation.

TABLE 10
ST TASK PARAMETERS

Task #	Deadline Time (ms)	WCET	Execution Time (ms)
T_1	14	0	0.000
T_2	14	0	0.0405
T_3	14	6	7.039
T_4	14	2	3.54
T_5	14	6	7.039
T_6	14	2	5.29

Ideally a p_{base} of $0.875ms$ is obtainable for configuration to enable the DYN messages get access to the FlexRay bus as frequently as possible. Due to hardware and software constraints this value could not be used. With 6 ST slots required at $40\mu s$ each this resulted in a ST segment size of $240\mu s$. With the cycles size of 1.750ms this leaves maximum 151ms to be divided between the DYN segment and the NIT. Each minislot in the DYN segment was set at 6MT. With the NIT of 25MT this resulted in a DYN segments size of $1485\mu s$ or 247 minislots. This is coupled with a worse case response time of 263 minislots.

TABLE 11
FLEXRAY FRAME PARAMETERS

Parameter	Value
Number of ST Slots	6
Number of DYN Slots	247
ST Slot Size	$40\ \mu s$
DYN Minislot Size	$6\ \mu s$
Payload Size	5 2-word-bytes
NIT	$25\ \mu s$

VI. CONCLUSION

This paper addresses the topic of migrating from CAN to FlexRay. This was carried out through the development of a generic migration framework. The migration framework involved synthesising tasks to the message level before obtaining associated FlexRay parameters. The proposed framework provides a solution utilising both the ST and DYN segments of the FlexRay cycle. The framework was then successfully implemented using ACC parameters. Experimental results show that the FlexRay parameters met previous CAN parameters, and also improved on them with the deterministic nature guaranteeing message transmission. The DYN messages were transmitted randomly so there were no predetermined timing constraints. These still demonstrate the use of both the ST and DYN segments available. Even with the improved results there are still large amounts of redundancy for use in future or larger applications. Therefore this method successfully migrates from a CAN to a FlexRay protocol.

VII. REFERENCES

1. Suri, V.C.a.N. *TTET: Event-Triggered Channels on a Time-Triggered Base*. in *Ninth IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age*. 2004.
2. Consortium, F., *The FlexRay Consortium Website*. 2008.
3. Cummings, R.W. *Easing the Transition of System Designs from CAN to FlexRay*. in *SAE World Congress & Exhibition 2008*. 2008. Detroit, MI, USA: SOCIETY OF AUTOMOTIVE ENGINEERS INC.
4. Roland Bacher, B.G.M., B.G.M. Herbert Haas, and I.A.W. Martin Simons. *Integration of FlexRay-based control units in existing test benches*. 2008 [cited; Available from: http://www.ixxat.com/article_flexray_gateway_feb08_en.html.
5. Jimmy Jessen Nielsen, H.-P.S.a.A.H. *Markov Chain-based Performance Evaluation of FlexRay Dynamic Segment*. 2007 [cited; Available from: http://rtn2007.loria.fr/5_Paper.pdf.
6. Microelectronics, F., *The Industry's Most Complete FlexRay Evaluation Kit Now Available from Fujitsu; Enables FlexRay Hardware Design, Verification Prior to Silicon*. 2005: http://www.fujitsu.com.
7. Shan Ding, N.M., Hiroyuki Tomiyama and Hiroaki Takada. *A GA-Based Scheduling Method for FlexRay Systems*. in *Proceedings of the 5th ACM international conference on Embedded software*. 2005. Jersey City, NJ, USA.
8. Aloul, N.K.a.F. *The Synthesis of Dependable Communication Networks for Automotive Systems*. in *SAE Internalional 2005*. 2005.
9. Traian Pop, P.P., Petru Elses, Zebo Peng, Alexandru Andrei, *Timinig Analysis of the FlexRay Communication Protocol*. 2006.
10. Schedl, D.A. *Goals and Architecture for FlexRay at BMW*. in *1st Vector FlexRay Symposium*. 2007. Stuttgart, Germany.
11. Eric Armengaud, A.S.a.M.H. *Automatic Parameter Identi cation in FlexRay based Automotive Communication Networks*. in *Emerging Technologies and Factory Automation, 2006*. 2006: IEEE.
12. Consortium, F., *FlexRay Communications System, Protocol Specification, Version 2.1, Revision A*. 2005. p. 245.