International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

# Implementing Flexray Controller for Flexray Data Logger

**Tukaram Tirvir[1], Supriya Shanbhag[2], Sonal Suryavanshi[3], Veeresh Ambe[4]**

[1, 2, 3] *Department of Electronics & Communication Engineering, KLS Gogte Institute of Technology, Belagavi, India*
[4] *Vayavya Labs Pvt. Ltd., Belagavi. India*

---

**Abstract -** *The work presented in this paper deals with the implementation of the Flexray controller on FPGA for Flexray data logger. The Flexray controller demonstrates the transmission of data from user (Graphical user interface (GUI)) to the Flexray bus, and also the reception of data from Flexray bus to the user (GUI). The design of the Flexray controller primarily involves the timing calculation so that the parameter values of flexray controller could be decided in accordance with the baudrate. The timing calculations largely affect the number of static slots to be used per cycle of Flexray, since this is vital in avoiding any overlap or loss of data between the number of data bytes transmitted or received per slot. An attempt is made to design a novel frame format for the frames received from Flexray bus, wherein timestamp of a received frame as well as direction of communication is declared. Two modules are designed viz. TX_reg and Tx_gui_module for storing and transmission of frames. Flexray controller implemented in the present work is able to test wakeup pattern and can also send the wakeup pattern and acts as coldstart node which can send startup frames. Results mentioned in the paper are simulation results in which reception part is carried out forcefully for demonstrating the working of reception.*

***Keywords:*** Flexray controller, Flexray frame, GUI, UART controller, wakeup, coldstart node, startup.

## 1. Introduction

As technology is advancing day by day, sports and luxury cars are using more and more sensors to meet the increasing demands. Number of ECUs used in the cars to provide automotive comfort to the customers has increased drastically than the ECUs that were used before. However, with comfort there comes complexity. Hence, as the modern cars have increased their comfort by increasing ECUs they face the hurdle of lack in speed of data transmission between the ECUs. This is mainly because they employ CAN bus system that provides 1 Mbps of speed which is not sufficient for vast amount of data. In order to overcome this problem, Flexray bus system was introduced by Flexray consortium.

Flexray is a time triggered protocol which has its own frame format [1] transmitted over a single slot. TDMA method is used by flexray protocol due to which each node has its predefined slot to transmit the frame. Each flexray controller has 64 communication cycles; each cycle has one static segment, one dynamic segment, one symbol window and a network idle time. Each static segment has 1024 slots i.e. 1024 frames. As it is time triggered protocol each node gets a fixed slot in which the node can transmit [1].

In a flexray bus system multiple nodes create a cluster in which values of some parameters must be same so that nodes can communicate with each other efficiently. To start communication in cluster at least a cluster must have two nodes which can send startup frames [1, 2] known as coldstart node. Flexray controller mentioned in this paper transmits two startup frames on different slots so that receiving node receives two unique startup frames and as a result it integrates with the cluster.

Flexray controller can wake up the channel according to user (GUI). Flexray controller is implemented on FPGA as FPGA provides better flexibility in terms of modifying the behavior of the Flexray controller. Implementing Flexray on FPGA also helps to test some extra features of Flexray driver which are not possible by general Flexray controller. Due to use of FPGA it is also possible to add other IP core along with Flexray IP core.

## 2. Related Work

Sergi Kosav has given a brief explanation on nodes, protocol operation controller, wakeup and startup in his "Flexray communication Protocol (wakeup and startup)" report [2]. Report states that to wake the channel a wakeup pattern comprising of a defined number of wakeup signals has to be transmitted. To start a cluster there must be at least 2 coldstart nodes which must transmit startup frame in different slot ids.

Jan Malinsky and Petr Kocourek implemented an education system which provides advanced features of flexray standard [3]. By using this system Flexray synchronization mechanism in operation was illustrated. System developed was suitable for training in Flexray technology. Main contribution through this system was that the system has the ability to influence the frequency of time base in order to test the SM behavior. Test is carried by using 10 nodes which are configured into a panel with one node acting as a motherboard.

Martin Patak has implemented Flexray controller on FPGA [4] which is able to work with Nios 2 bus processor which has 32 bit parallel interference. New commands to wake the channel and to start the cluster were implemented in this work. Flexray controller implemented was able to transmit wakeup signal as well as act as a

coldstart node. This controller can also test some extra features like wakeup which is not possible by general flexray controller.

## 3. Methodology

### 3.1 Project requirement

Flexray controller is implemented on Xilinx ZC702 FPGA board; to transmit and receive data from Flexray bus transceiver is used. Coding is carried by using VHDL language on Vivado IDE. Logic analyzer along with PulseView is used for testing and debugging of signals.

### 3.2 Project Design

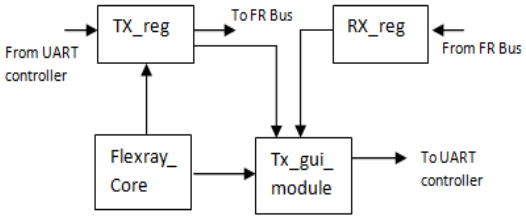



Fig. 1: Modular design

Figure 1 show modules required for flexray controller to implement on FPGA for Flexray data logger.

All the frame entries received from UART controller which are to be transmitted are first stored in TX_reg. After storing all the entries TX_reg then transmits them according to slot id after transmitting wakeup and startup signal onto Flexray bus. This module also gets some informative signals from the Flexray core.

Flexray core is the heart of the system where all the execution of command takes place, core provides macrotick and cycle count with the help of which TX_reg can transmit the data.

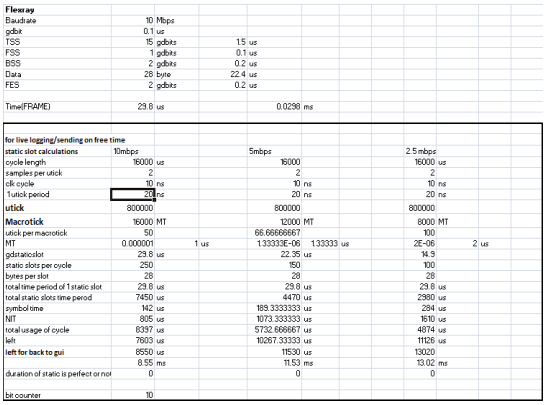
RX_reg is module which is responsible for receiving Flexray Frame from Flexray bus and giving it to Tx_gui_module.

Tx_gui_module stores the acknowledgement of transmitted frame and the Received frame so that it can be sent back to GUI via UART controller for display. This module also stores status.

### 3.3 Contribution to the project

#### Timing calculation

Figure 2 shows the timing calculation which helps to determine max no. of slots which can be used as well as max number of bytes per slot that are allowed.



| Flexray | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Baudrate | 10 | Mbps | | | | | | |
| gdbit | 0.1 | us | | | | | | |
| TSS | 15 | gdbits | 1.5 | us | | | | |
| FSS | 1 | gdbits | 0.1 | us | | | | |
| BSS | 2 | gdbits | 0.2 | us | | | | |
| Data | 28 | byte | 22.4 | us | | | | |
| FES | 2 | gdbits | 0.2 | us | | | | |
| Time(FRAME) | 29.6 | us | | | 0.0296 | ms | | |
| for live logging/sending on free time | | | | | | | | |
| static slot calculations | 10mbps | | | | 5mbps | | 2.5 mbps | |
| cycle length | 16000 | us | | | 16000 | | 16000 | us |
| samples per utick | 2 | | | | 2 | | 2 | |
| clk cycle | 10 | ns | | | 10 | ns | 10 | ns |
| 1 utick period | 20 | ns | | | 20 | ns | 20 | ns |
| utick | 800000 | | | | 800000 | | 800000 | |
| Macrotick | 16000 | MT | | | 12000 | MT | 8000 | MT |
| utick per macrotick | 50 | | | | 66.66666667 | | 100 | |
| MT | 0.000001 | | 1 | us | 1.33333E-06 | 1.33333 | 2E-06 | 2 us |
| gdstaticslot | 29.8 | us | | | 22.35 | us | 14.9 | |
| static slots per cycle | 250 | | | | 150 | | 100 | |
| bytes per slot | 28 | | | | 28 | | 28 | |
| total time period of 1 static slot | 29.6 | us | | | 29.6 | us | 29.8 | us |
| total static slots time period | 7450 | us | | | 4470 | us | 2980 | us |
| symbol time | 142 | us | | | 189.3333333 | us | 284 | us |
| NIT | 805 | us | | | 1073.333333 | us | 1610 | us |
| total usage of cycle | 8397 | us | | | 5732.666667 | us | 4674 | us |
| left | 7693 | us | | | 10267.33333 | us | 11126 | us |
| left for back to gui | 8550 | us | | | 11530 | us | 13020 | |
| | 8.55 | ms | | | 11.53 | ms | 13.02 | ms |
| duration of static is perfect or not | 0 | | | | 0 | | 0 | |
| bit counter | 10 | | | | | | | |

Fig. 2: Timing calculation

#### Frame format

Frame format shown in Table 1 is for data which is received from Flexray Bus.

| R | Sync Frame | Startup frame | Frame ID | PL | Cycle count | Time Stamp | Data |
|---|---|---|---|---|---|---|---|

Table 1: Frame format

| | |
|---|---|
| 'R' | - direction (1 Byte) |
| Sync Frame | - 1 bit |
| Startup frame | - 1 bit |
| Frame ID | - Frame ID (8 bits) |
| PL | - Data Length (5 bits) |
| Cycle count | - Cycle Count (6 bit) |
| Time Stamp | - 43 bits |
| Data | - Data |
| Total (Bytes) | - 9bytes + Data bytes |

#### Modules

### 1. TX_reg

Data from UART controller is stored in this register and processed. Transmission of frame is also carried out by this module. This module has more than one process out of which one process is storage and start of transmission which is shown and explained in figure 3 with the help of state diagram.

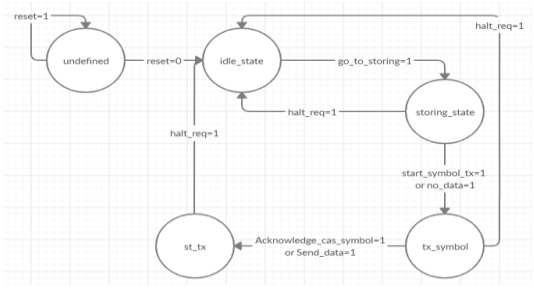



Fig. 3: TX_reg

As shown in figure 3 process shifts to idle state, when reset is set to 0. There is a shift of state from idle to storing

when process gets go_to_storing which is triggered from Flexray core. After storing is completed, state shifts to tx_symbol where the module transmits the symbol (wakeup and CAS). After transmission of symbol the state shifts to st_tx where actual frame is transmitted. In any state if the module gets the halt_req it shifts to idle state.

## 2. Tx_gui_module

Whenever a frame is transmitted a back acknowledgement has to be transmitted to the GUI. The acknowledgement of transmitted frame and received frames have to be transmitted to GUI once the static segment is finished so due to that the acknowledgement as well as the received frame has to be stored somewhere. Hence the module has a storing process which is explained in figure 4 with the help of state machine

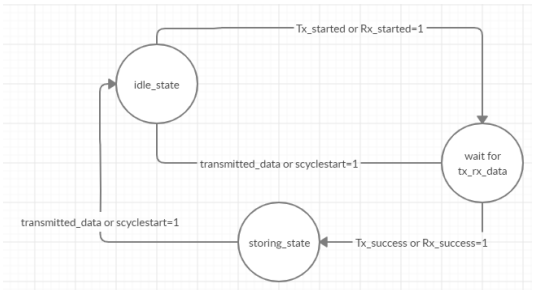



Fig. 4: Tx_gui_module

Whenever transmission or reception of frame is started, process changes from idle to wait_for_tx_rx_data. When there is success of frame transmission or reception process shifts to storing state where it stores the acknowledgement of transmitted frame and received frame. Process shift to idle state when data is transmitted back to GUI that is transmitted_data signal is triggered or scyclestart is triggered

Module also has one more process which stores the status of POC, wakeup, and startup of the POC module along with timestamp. Storing is carried by using case statement. As soon as static slot is finished the process trigger st_tx_gui this indicates UART controller to transmit the data to GUI.

### 3.4 Problems Faced

The error displaying mismatch registers length message occurs at synthesis level. This happens when we try to copy data from one register of specific length viz. 2 bytes to another register of length more than or less than 2 bytes. Another error occurs at implementation level called the Net overdriven problem; all the registers in FPGA are connected with lines known as Nets. When a single Net gets the data from two different register at a time with different logic (0 or 1) than the Net gets overdriven by multiple logic due to which the output becomes undefined in simulation and in message window we get Net overdriven message.

## 4. Results

The results obtained are presented in two parts. The first part deals with transmission of wakeup, startup and frame while the second part deals with reception of frame from Flexray bus. As the results are derived by simulation, reception of frame is obtained forcefully. Actually transmission and reception of frame on same slot id is not possible if a node is transmitting wakeup pattern or CAS symbol and if it receives wakeup pattern or CAS symbol on same channel then transmission is stopped. So the motive behind receiving frame, wakeup pattern and CAS symbol forcefully is to just demonstrate the working of reception module.

1. Wakeup pattern, CAS symbol and frame transmission

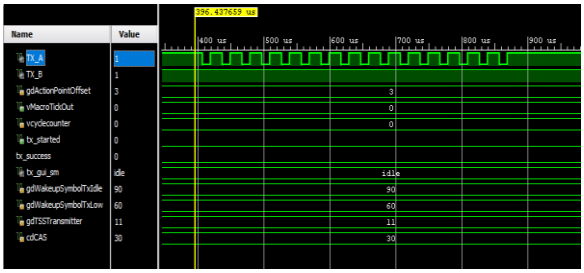



Fig. 5: WUP pattern

Figure 5 shows the transmission of wakeup pattern which has multiple wakeup symbols when send_wakeup_req is triggered. Wakeup pattern is used to wakeup channel A or B for communication. Wakeup symbol is a signal which is idle for some period and then low for some period. The period of idle and low is decided according to the baudrate. One Wakeup symbol of wakeup pattern shown in fig 5 is 90*gdbit idle and 60*gdbit low. gdbit = 0.2 us.

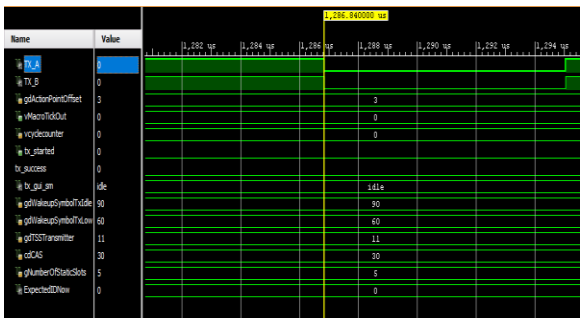



Fig. 6: CAS symbol

As soon as wakeup Pattern is transmitted CAS is transmitted to start the communication. CAS symbol is a signal which is low for some particular time period. Again this period is decided on baudrate. CAS symbol is calculated by adding 2 term; gdtsstransmitter which is 11gdbit and cdCAS which is 30gdbit. So CAS symbol shown in figure 6 is low for 41gdbit

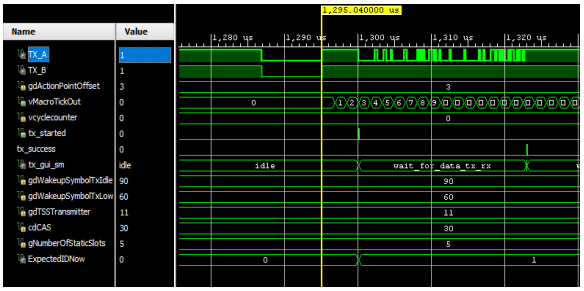


Fig. 7: Frame transmission

As shown in figure 7 after CAS symbol is transmitted, controller transmits startup frame in a given slot id. In frame shown in above fig has slot id 1. The controller determines the respective slot in which the frame has to be transmitted by macrotick which starts running as soon as CAS symbol is transmitted. Each frame gets its macrotick (gdActionpointoffset) number with respect to its frame id. When the respective macrotick arrives frame is transmitted. In Flexray there are 64 cycles in each cycle the number of slots can be decided by user, in this case the number of slots per cycle is 5, which means 5 frames can be transmitted in one cycle. To start a cluster, a controller must transmit 1 frame for 4 cycles and then after 4 cycles 2 frames with different slot ids for rest of the cycles. After 8 cycles another node gets integrated with this node and the cluster starts.

2. Wakeup pattern, CAS symbol and Frame reception

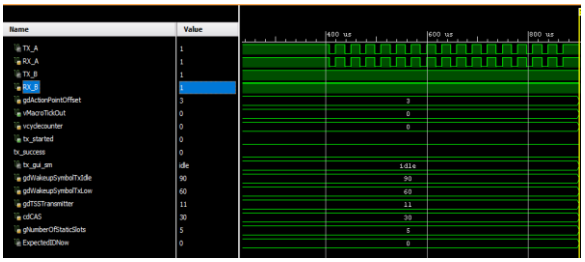


Fig. 8: WUP pattern

As shown in figure 8 Wake pattern is received on RX_A pin. System will only receive the wakeup pattern when idle and low period of wakeup symbol is same at both the ends, if not then the system will not recognize the wakeup pattern.

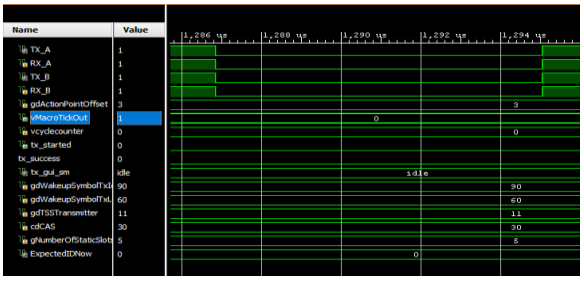


Fig. 9: CAS symbol

As shown in figure 9 CAS symbol is received on both channels that is RX_A and RX_B. In this case also system is able to recognize CAS symbol when the low period of CAS symbol is same at both the ends.

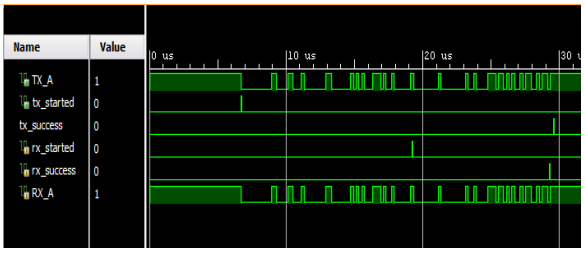


Fig. 10: Frame transmission and reception

After 8 cycles, as soon as another node gets integrated with our node, communication starts. Data frames can be received as well as transmitted as shown in figure 10. Frame id used by our node cannot be used by another node.

## 5. Conclusion and Future work

Previous work carried on Flexray provides a system for testing and maintaining the synchronization of clock [3]. Another work [4] describes a system built to work as a Flexray logger; however it fails to work as more than one node. The system described in the present work addresses this problem and is able to work as more than one node. It helps to perform some basic testing such as wakeup pattern and is able to work as coldstart node which can send startup frames and also be able to work as logger which can read and write data from/to the flexray bus. The system works in static segment where data has to be first stored and then is transmitted repeatedly. Further work can be extended to modify this system in such a way that it also works in the dynamic segment where data is transmitted as soon GUI transmits.

## References

1. FlexRay Consortium, FlexRay Protocol Specification V2.1 Rev.A, 2005.
2. Sergi Kosav, “Flexray communication Protocol (wakeup and startup)”.
3. Jan Malinsky, Petr Kocourek, “Demonstrational system for training in flexray communication”, XIX IMEKO World Congres, Fundamental and Applied Metrology, Lisbon, Portugal, September 6_11, 2009.
4. Martin Patak, “FlexRay controller”, Prague, 2012.