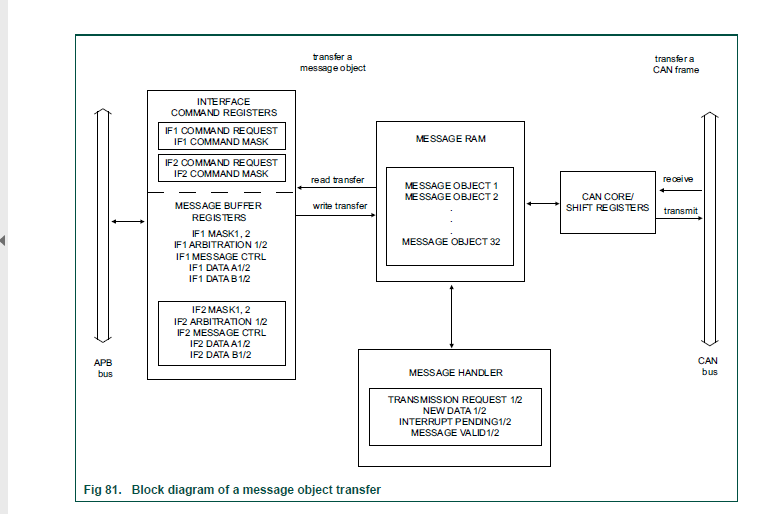
LPC1549的CAN 控制器的相关说明

Controller Area NetWork 结构框图



这里有 Message Interface 寄存器和 APB 总线相接，负责和CPU进行运行时数据交互，

Message RAM（拥有32个Message Object） 负责和 Message interface 交互，同时负责和CAN\_CORE/SHIFT REGISTERS 交互。而CAN\_CORE/SHIFT REGISTERS 负责和CAN bus 交互。寄存器简要说明见另一篇文档。

# 一：初始化CAN控制器

每次reset 后，**CAN控制器**的**大部分寄存器**都保持**不变**，CAN控制器的 CAN control register 的 **INIT** 被置位为 1，**使能CAN控制器**的**软件初始化**。并且 busoff 位被置零， CAN\_TXD为高电平。由于使能了软件初始化，这个时候CPU可以对CAN 控制器进行初始化。

每当 INIT 被置位时，就可以对CAN控制器进行软件初始化，初始化期间**暂停和CAN BUS的通信**，**CAN\_TXD** 引脚为**高电平**，EC（error counter）**不计数**，配置寄存器保持不变，如果**CCE位被置高**，bit timing register 和 BRP extension register 就可以**被读写**修改。配置步奏，先要配置 bit timing register，然后配置 message object。

当设置完毕后，通过把 INIT 位清零，BSP 同步后，等待 CAN\_TXD 的 11 位连续高电平后（CAN BUS 空闲），接入总线，开始进行数据传输。

# 二：配置 Message Objects（Interface –> Message）

配置Message Object ，配置该片寄存器时，只需要把MSAGVAL 设置为 valid。在初始化时配置 Message Object ，需要把所有的Message Object 配置成特定格式，或者都把MSGVAL设成为0 （not valid）**。配置 Message Object** 是通过**配置两个** **interface register** 之一的**MSK1/2，ARB1/2，MCTL，DA1/2,DB1/2**来完成的。**配置完这些数据后**，通过**写 IFx Command Request Register（IF1/2\_CMDRQST）** 把 **IFx Message Buffer Registers** 中数据**同步**到Message RAM中特定（**由IF1/2\_CMDRQST中NM位指定**）的 Message Object 里。

# 三：控制器运行

**数据接受：**

一旦初始化完成后，CAN 控制器会自己接上 can bus。接受到数据后，如果数据通过了 Message Handler’s 的过滤处理，数据会被存在合适的 Message Object 中。CPU 可以通过 Message Interface Registers 读取该些数据。

**数据发送：**

等待传送的数据由CPU准备好，如果数据的特定filed存在一个**配置好的特定**的（arbitration，控制位在**初始化期间**配置过）Message Object，那么只有数据 filed 会被传输过去，然后 **TXRQUT 和 NEWDAT** 位都会被置位，然后**开始传输**过程。

# 四：TXRQST 和 NEWDAT

一旦开始发送，**TXRQST 就会被清零**，只要 TXRQST 是零，就不允许再次发送。然而 NEWDAT 保持置位直到数据发送完毕后，清零。所以**当一次发送失败**（失去 **arbitration或者发生错误**）时，**NEWDAT**仍然保持为高，这时候**为了重启发送**，需要**再次置位 TXRQST**。

TXRQST = 1 意味着有数据要发送，TXRQST = 0**表示正在发送**或者**阻止发送**。

NEWDAT = 1 意味着有新数据到来，或者新数据没有被处理完毕，NEWDAT = 0 表示数据已经被处理完毕。

# 五：CAN message handler

CAN message handler 负责处理CAN\_CORE 和 RX/TX Shift Register之间的数据传输过程，还有 Message RAM 和 IFX Registers 的数据传输过程。有如下四种功能。

1：从 Message RAM传输数据到 shift registers

2：从 shift registers传输数据到 Message RAM。

3：在 IFx Registers 和 Message RAM 之间传输数据。

4：从 shift registers 传输数据到数据过滤单元。

5：扫描 Message RAM 中对象以匹配特定 Message Object。

6：处理 TXRQST 标志。

7：处理中断。

一旦 INIT 位被置位，那么**CAN 控制器的状态机**和 **Message Handler** **状态机**控制着整个 CAN controller 内部的**数据流**。接受**通过过滤器的数据**，发送数据。CPU 通过 IFx Interface Registers 读取接受到的数据和发送数据。

# 六：在IFx Registers 和 Message RAM（包含32个Message Object） 传输数据

当cpu 准备好在 IFx Registers 和 Message RAM 直接传数据时， Message Handler 会把对应的CMD 寄存器的 BUSY 位设置为 1。当传输完毕后再重置为 0。由于 Message RAM 的结构原因，不能够对 Message Object 进行单独的位或者字节传输，每次传输只能够写整个 Message Object，所以为了只修改某些位，而不影响其他位，只能够进行 read – modify – write 这样的步奏，才能完成数据传输。

先利用 CMDMASK 寄存器读出不需修改的 Message Object，这里没有被 CMDMASK 选中的将会保持不变。(**这里的读是通过设置 写IFx\_CMDREQST**)

然后把 Message buffer 中数据写入 Message Object，此时没有被 CMDMASK 的部分会保持Message Object 中本来的数据。

# 七：shift register 和 CAN\_CORE 之间的数据传输

**发送：**

只有当 **IFx\_register** 和 **Message RAM** 之间没有数据传输，而且shift register 准比好接受数据时，并且有 Message Object 的 TXRQST 还有 MSGVAL 都为有效值时，那么这些Message Object中优先级最高的会被载入 shift register，准备发送。

**接收：**

每当一个CAN消息的ID场和控制场都被接收到 **RX 的shift registe**r 中，Message Handler便会搜索有效的，匹配的 Message Object。为了去搜索匹配的Message Object，

Acceptance Filtering unit 会**从 RX shift register** 中**载入 arbitration bits**，然后再把Message Object 中**的arbitration ，mask filed 载入 Acceptance Filtering unit**，进行**匹配**，如果**匹配成功**后，会**根据消息的类型**进行处理（数据帧，远程帧），如果扫描到边界都没有匹配则结束扫描。忽略信息。

# 八：CAN Message RAM接受消息帧：

**1：数据帧**：

每当把接受的数据从 shift register 传输到Message Object 时（也就是Message RAM 接收到Message Object），Message Handler会把仲裁场，数据长度场，数据**存入相应的 Message Object**。每当Message Object 接收到数据后都会把 **NEWDAT 位置位1**。如果Message Object 的**NEWDAT 之前就为 1**，那么就会**触发MSGLST （消息丢失）位为1**。在接受数据后，该 Message Object的**TXRQST位为0**，阻止传输数据。

**2：远程帧**：

1. 每当Message RAM接受到远程数据帧时，如果对应 Message Object中的 DIR = 1，RMTEN = 1， UMASK = 1 or 0：

对应的Message Object 的 **TXRQST 位被置位 1**，并且数据会存在 shift regist中，数据不会存入Message Object变，等待**转发该帧**。

1. DIR = 1，RMTEN = 0，UMASK = 0：

不修改对应的TXRQST，**忽略**该帧，不转发。

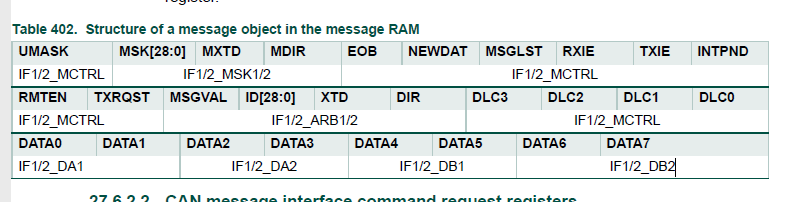
1. DIR = 1，RMTEN = 0，UMASK = 1：

接收到该帧后，该Message Object的**TXRQST 被清零**，然后把帧存入 Message Object 并且 **NEWDAT 被置位**，说明有新数据需要处理。

**3：接受和传输优先级**

Message Object 1 优先级最高，Message Object优先级最低

# 九：Message Object Structure

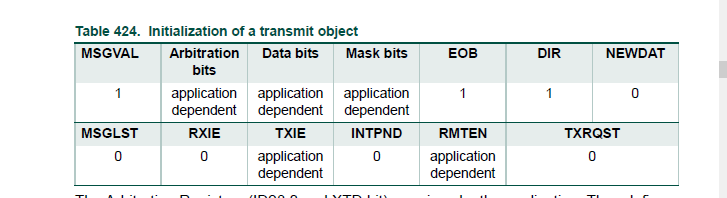


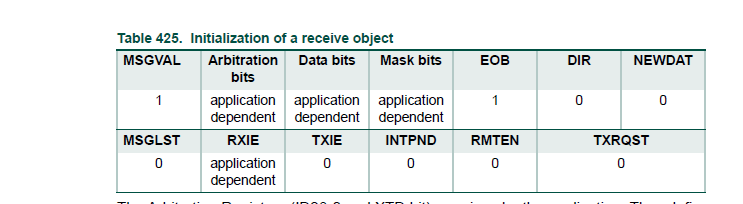
这里的**每个标题栏下方**都有一个 **Interface register** 的相应寄存器，这表示的是**这些值**是通过 interface register 中**某个寄存器**的**相应的同名位**来设置。比如 UMASK 下方是 IF1/2\_MCTRL 表示的是该位由 **IF1/2\_MCTRL** 中**相应UMASK位**去设置。再比如 MSK[28:0],就是由 IF1/2\_MSK1/2 中的 MSK[28:0] 来设置。

# 十：初始化和更新transmit object 和 receive object（其实就是数据发送和接收）

**初始化：**

Transmit object 的初始化需要按照下列格式来，其他未列出的都是保持不变。这些位的改变都是通过设置对应的IFx registers，然后CPU 写 **CMDRQST**寄存器，把数据更新到对应的Message object 中。然而**对于receive object也需要配置一次**。





更新：

更新transmit object，如果**只更新数据**，则需要先向 CMDMSK（command mask register） 去写入 0x0087。然后写 CMDRQST （command request register）。

读入 receive object ，先向CMDMSK 写入0x007f（设为写方向，屏蔽其他位，只读数据），然后写 CMDRQST。

# 十一：FIFO(其实就是把Message object 连接起来)：

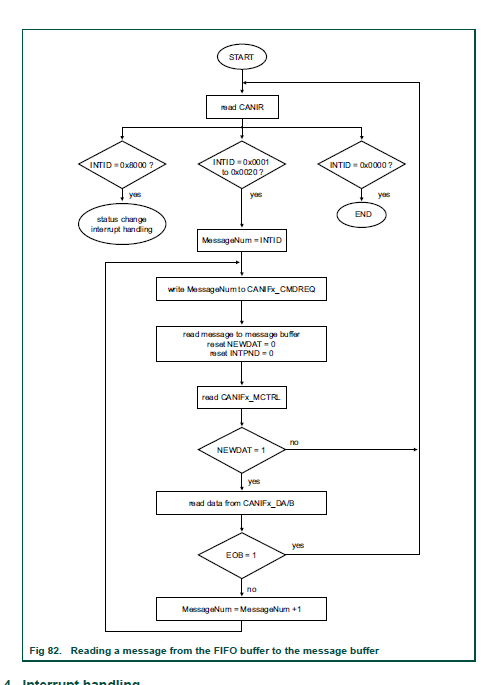
**配置 FIFO Buffer：**

FIFO buffer 也包含了Message object ，除了要多写一位EOB位外，配置FIFO 的Message Object 和配置 Message RAM 的receive Object 是一样的流程。

为了拼接两个或者多个Message Object 的FIFO Buffer，这些Message Object都需要把ID 和 mask 配置到匹配的值。优先级最高的Message Object 会成为第一个FIFO Message Buffer。另外FIFO Buffer 的最后一个Message Object 的EOB 位必须置位 1。

**FIFO接受Message：**

匹配FIFO的消息会被存入到FIFO Buffer，并且Message Object 的NEWDAT 会被置位。如果此时该Message Object的EOB 是0，那么该Message Object 会被锁定，不允许再写入直到他的 NEWDAT 被清零。另外如果FIFO 中所有的Message Buffer 都没有被读取，那么后续的要存入FIFO的消息都会被写入最后一个Message Object。



# 十二：CAN 中断处理：

**状态中断**优先级最高的，**消息中断**的优先级和消息对象（Message Object）的序号有关，越低越高。**消息的中断**的**清除**可以把 **Message Object的INTPND 清零（先设置CAN memory control register，然后设置）**，而Status Interrupt 通过**读取 Status Register** 来**清除**。Interrupt register 中的INTID 指示了中断源。

如果IE 使能，那么硬件会通知CPU有中断发生，中断信号一直保持为高，直到Interrupt Register 被清零（读Status Register或者清除Message Object 的INTPND）。

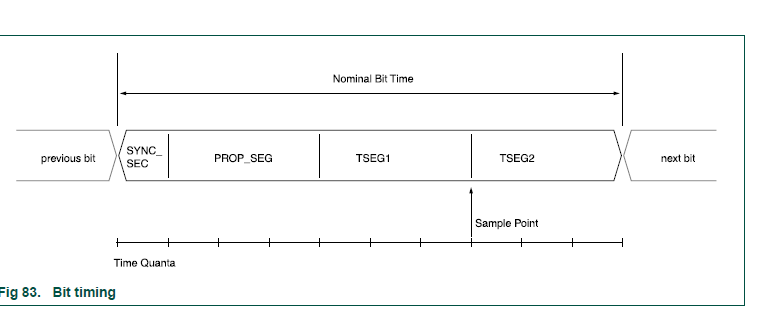
**Interrupt Register** 的值为**0x8000**，指示这是状态中断（CAN 核心改变了Status Register），如果为其他值，则为消息中断。**CPU**可以对 Status Register **进行读写**，但是却**不会产生中断**，也**不会清除**中断。（**只有读**可以**清除Status中断**）

其他值指示的消息中断，INTID指示了产生消息中断的最高优先级Message Object。

CPU 通过IE决定**中断产生后**是否通知CPU（**使能Interrupt Line与否**），即使IE未使能，但是Status Register的状态还是会更新。如果想不更新，通过EIE和SIE来确定。

对于**消息中断**，CPU可以通过**读 Interrupt Register**的**INTID** 来知道中断源，也可以**读INTPND** 来**获得消息中断**的中断源。

# 十三：bit timing ：



CAN 总线的单个位的时间为：

T\_BT = SYNC\_SEC（） + PROP\_SEG + TSEG1 + TSEG2

他们的含义分别为CAN 时钟**单个周期长度**，**时延长度**，**采样点前时间段**，**采样点后时间段**，这些时间的长短都是**单个周期**长度的**整数倍**。。

所以波特率为 f = 1/T\_BT = 1/(n\*)。