在 30 分钟或更短的时间内开发 CAN

作者：Hassane El-Khoury
业务开发经理
Cypress 半导体公司

摘要

自从上世纪 80 年代引入 CAN 以来，在其规范和需求方面出现了巨大演变。由于其扩展能力，使得其在众多应用中得以广泛使用，从汽车到工业机器和工业自动化。随着其广泛使用，实施复杂性也在两个层面上相应增加：

- CAN 控制器设计从基本控制器转向全 CAN 控制器，在某些情况下，甚至为扩展的全 CAN 控制器。
- CAN 软件堆栈从汽车通信堆栈变为 CANOpen，以及 DeviceNet。

假定 CAN 仅是汽车系统中的单个组件，开发人员仅需面对尽可能少的挑战就能实现它，那么就能够将工作重点放在系统级功能上面，而不是纠缠于外围配置。在本文中，将研究 CAN 接口，讨论不同的实施、配置方式，以及调试接口以简化设计。

控制器区域网络（CAN）最早是由 Robert Bosch 引入的，用于处理不断增加的车辆功能和网络复杂性。在嵌入式系统开发的早期阶段，模块中包含单个 MCU，可执行单个或多个简单功能，如通过 ADC 读取传感器电平信号并控制直流电机等。随着这些功能日益复杂，设计人员采用分布式模块架构，在同一 PCB（印刷电路板）的 2 个或多个 MCU 中实现其功能，并使用 I2C 或 SPI 协议执行这些功能之间的通讯。对于前面的相同示例，综合模块则具有用于执行所有系统功能、诊断和故障保证的主 MCU，同时另一 MCU 负责 BLDC 电机控制功能。以较低的成本采用广泛使用的通用 MCU 即可实现该目的。

在当前的车辆中，功能分布于车辆中而不仅仅是模块内，模块间的通信协议需要高的故障容差，，因而在汽车市场中设计并引入了 CAN。

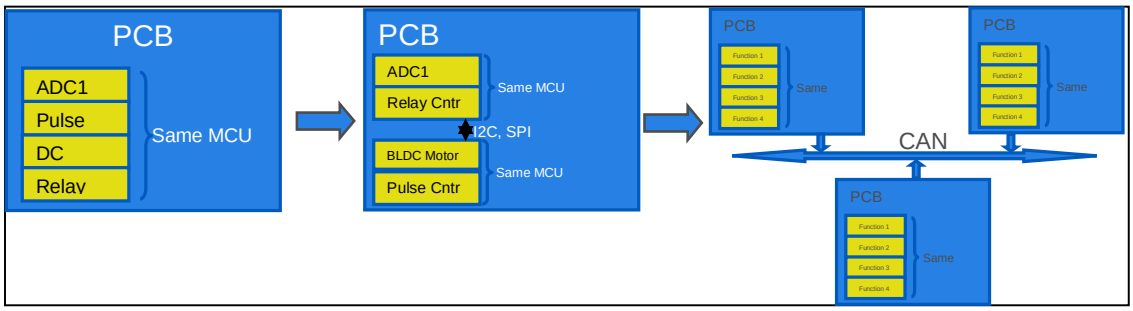

图 1 - CAN 介绍

到上世纪 90 年代中，随着 DeviceNet 和 CANOpen 协议的引入，CAN 以及工业控制已在汽车之外的范围得到广泛应用。

受市场中这类需求的推动，很多 MCU 供应商内部集成了 CAN 控制器，用于推广这类市场。虽然在高层面上 CAN 的功能看上去类似于 I2C 或 SPI，允许两个节点之间的通信，但在控制器层面上 CAN 通信与他们存在本质不同，下面列出 CAN 能提供的服务：

- CAN 通信基于消息，而不是基于地址。
- CAN 节点能够在总线上发送或请求消息。
- 复杂的错误处理机制。
- 使用 CRC-15（循环冗余校验）保护，能够探测一行中出现的 5 个损坏位。

由于是基于消息而不是基于地址，CAN 总线上的节点可拥有多条传输的消息，也就是说，制动器模块可能具有包含车辆速度信息的消息，包含传感器信息的消息（如轮速传感器），以及包含诊断信息的消息，后者具有最高的传输优先级。

首先考察节点中的消息优先级和消息 ID 解码，人们或许认为 CAN 会使 CPU 产生高的负荷，从而阻止更多复杂功能的集成。这类问题是可以通过不同类型的 CAN 控制器解释，如图 2 所示。

- 基本 CAN 控制器：在 CAN 控制器硬件中实现了十分基本的过滤功能，减少了消息处理，具有较高的 CPU 负荷。在基本 CAN 控制器中，CPU 会从 CAN 控制器收到多条中断信号，以便接收、确认和分析消息，也会从应用程序一侧收到多条中断信号，以便确定是否要根据所收到消息的 ID 传输应答。基本 CAN 控制器仅应在低波

特率、低消息通信数量的情形下使用，从而使得 CPU 能够处理额外的非通信任务。

- 全 CAN 控制器：提供了扩展的消息过滤功能，以及硬件消息分析功能，解放了 CPU，使其无需再对每一接收到的消息做出回应。可对全 CAN 控制器进行配置，仅当消息 ID 已在控制器中设为接受时，才会中断 CPU。全 CAN 控制器还设置了多个消息对象，相当于邮箱，邮箱能够储存特定的消息信息，如收到的 ID 和数据字节，以供 CPU 检索。在该情形下，CPU 可在任何时间检索消息，但必须在更新所收到的相同消息并改写邮箱当前内容之前完成任务。这类情形在最终类型的 CAN 控制器中得到了解决。
- 扩展的全 CAN 控制器：通过提供用于接收消息的硬件 FIFO，提供了额外层面的硬件实施功能。通过这类实施方案，在中断 CPU 中断之前，可保存相同消息一个以上的情况，因而防止了高频率消息的丢失，通过该功能，CPU 甚至能够在较长的时间段内专注于主要的模块功能。

请注意，DeviceNet 将过滤判据扩展到了 ID 字段之外，扩展到数据的前两个字节，在实施该协议时，必须使用全 CAN 控制器或扩展的全 CAN 控制器。

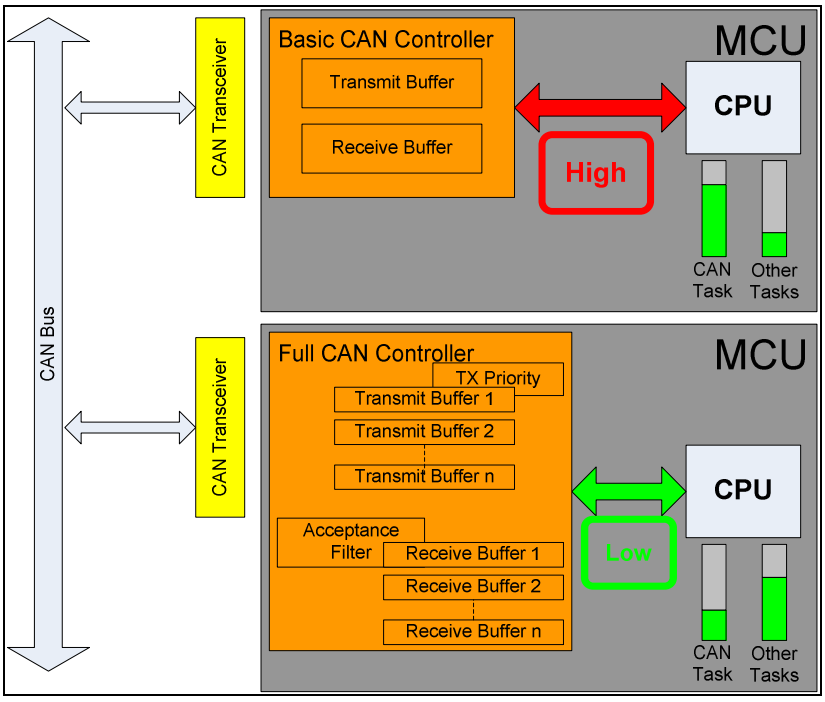

图 2 – 基本 CAN 控制器和全 CAN 控制器

根据具体的消息结构，上述两种配置可共存于单个模块中，以便实现高的消息优先级，并改善 CPU 对所收到消息的处理。例如，对于接收一条消息（如 ID = 0x250）中实效保护信息以及另一消息（如 ID = 0x3FF）中温度传感器信息的模块，可将第一个 CAN 控制器配置为全 CAN 控制器，并将第 2 个配置为具有 4 缓冲 FIFO 的扩展全 CAN 控制器：收到每条失效保护消息时，以及温度传感器消息时，CPU 被中断。在图 3 中，给出了这类 CAN 控制器的配置图，它具有一个可定制的虚拟 CAN 控制器，能够快速实现复杂的消息处理，这三种类型的 CAN 控制器可共存于其中：

- 消息 5 → 基本 CAN 邮箱。
- 消息 0x250 → 全 CAN 邮箱。
- 消息 0x3FF → 扩展的全 CAN 邮箱。

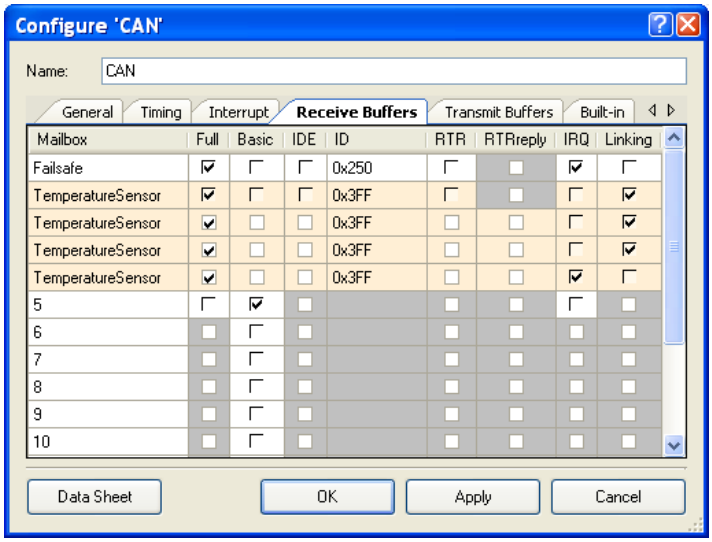

图 3 - PSoC Creator - CAN 控制器配置

除了其功能方面的优势外，CAN 还因其高的容错性而得到广泛应用。其比特率高达 1Mbps、总线长度可达 1000 米（在 50Kbps 下），因而必须遵照 CAN 比特时序，以便在电子噪音环境下工作，同时保持高水平的故障检测和校错特性。

为了确保高水平的容错性能，与 CAN 一起引入了亚比特时序结构，能够实现更严格的控制，确保每一 CAN 总线的正确总线状态。

一个单独的 CAN 位由 4 个段表示：

- **Sync_Seg**：用于同步总线上的各种节点。
- **Prop_Seg**：对物理延迟进行补偿（物理总线和内部 CAN 节点上的传播延迟）。
- **Phase_Seg1，Phase_Seg2**：用于补偿相位边缘误差。在在同步过程中，会缩短或加长这些段。

仅具有 3 个段的 CAN 控制器也很常见，其中，Prop_Seg 添加到了 Phase_Seg1 上。

在图 4 中，显示了位时序表征，以及实现它所需要的所有参数：

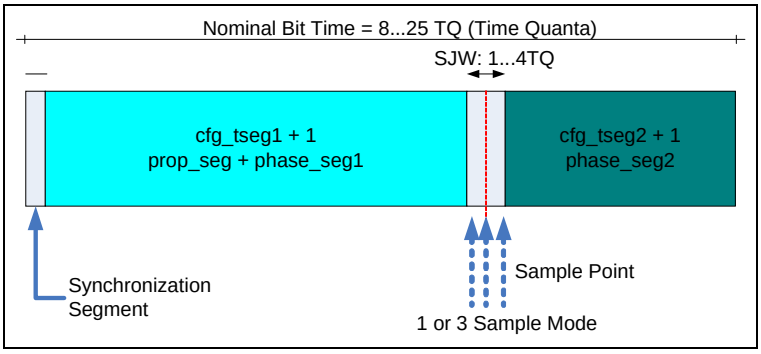

图 4 – 位时序表征

请注意，所有的 CAN 位时序计算均基于时间量（TQ），它定义为固定的时间单位，由振荡器导出，取值介于 8 和 25 之间。按照时间方面的观点，1 个 TQ 等效于 1Mbps 总线速度下 1 微秒位长度的 1/25th 位或 40 纳秒。

根据 CAN 位时序的过度简化规则，可进行下述设置以管理 4 比特时间段的值：

- Sync_Seg = 1TQ
- 1TQ ≤ Prop_Seg ≤ 8TQ
- 1TQ ≤ Phase_Seg1 ≤ 8TQ
- 2TQ ≤ Phase_Seg2 ≤ 8TQ
- 1TQ ≤ SJW ≤ MIN(Phase_Seg1，4)
- SJW：同步跳跃宽度，它定义为时间长度，可加长 Phase_Seg1 并缩短 Phase_Seg2。

上述关系会为每一包含的参数生成大量的可能值，要想成功实现可靠的 CAN 通信，选择正确的组合至关重要。设计人员不仅必须考虑振荡器的精度、所涉及节点内的传播延迟，还需考虑必须与之建立通信的其他系统节点。

根据系统时钟、所需的波特率、以及可能要求的位样本点（如，在 87.5%处要求的 CANOpen 样本点），为 CAN 设定位时序成为很多设计人员不愿意面对的挑战。这种复杂性导致很多新的嵌入式系统再次使用传统的 MCU 甚至软件堆栈，而不是采用能够以更好方式处理整个系统要求的新产品。不幸的是，这类情形使得 CAN 被归于嵌入式外围装置的“它能工作，但我不想接触它”类别下。

在图 5 中，显示了 CAN 的位时序结构示例，其中，对于正在工作的 CAN，创建新的时序分析或更改已有节点结构的波特率不再是有危险的任务。通过提供所有指定并验证的参数组合，可实现可靠的 CAN 时序，设计人员能够将重心放在模块更复杂的主要功能任务方面。

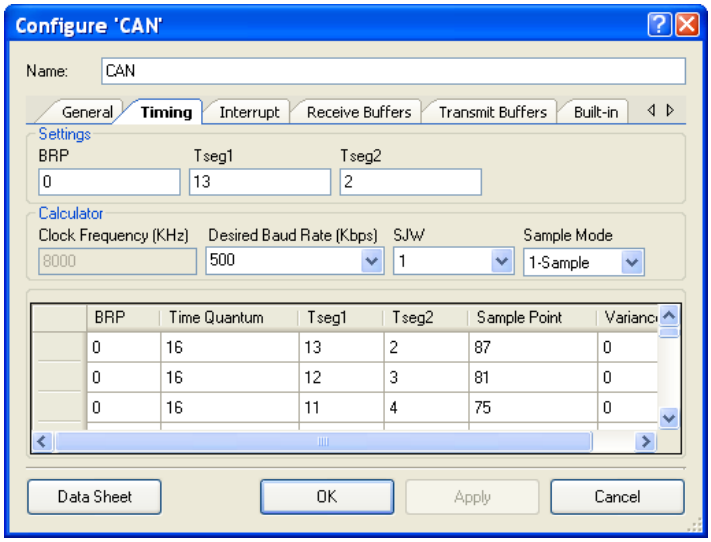

图 5 – PSoC Creator – CAN 位时序结构

尽管它出现的时间已有十多年，但 CAN 仍被认为是嵌入式设计中的复杂外围装置，尤其是研发领域更是如此。研发活动涉及系统配置的多次迭代，CAN 为整个系统中的一个部件。可将 CAN 驱动和 CAN 堆栈开发外包给专业公司，这类公司收取一定费用即可提供相应服务，某些公司在交付后更改和配置服务时还要另收费。近来，某些半导体公司（如上述示例中的 Cypress）提供了多种弥补这部分不足的工具，使用这些工具，可进行快速的内部开发，更快地推向复杂嵌入式系统市场。