

草稿  
Draft

# femto<sup>basic</sup>

## 兼容 RISC-V 指令的超轻量级 MCU 软核

A super light-weight MCU soft core compatible with RISC-V instruction set

[ricyn@foxmail.com](mailto:ricyn@foxmail.com)

2021-10-16

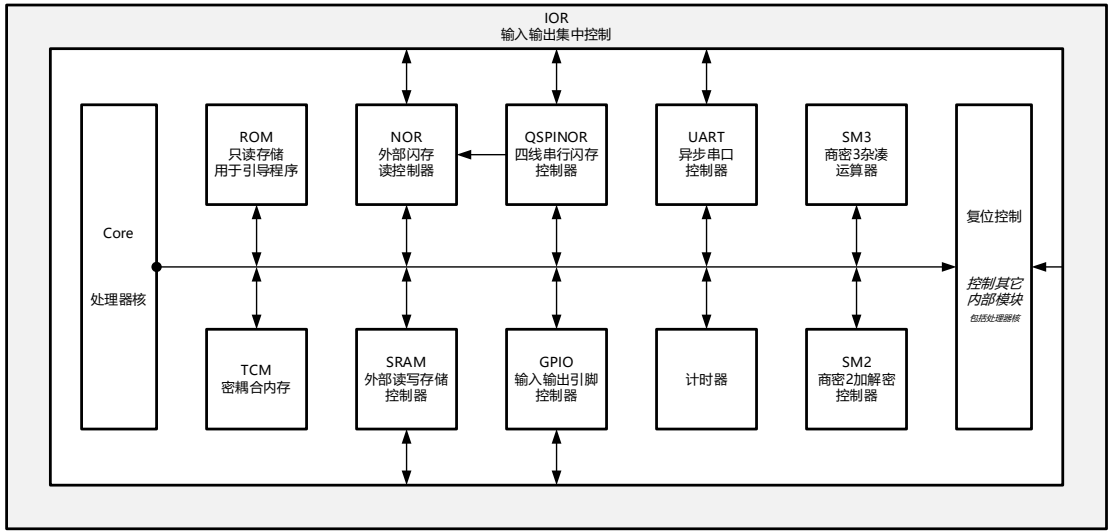
# 概览

femto 是一款兼容 RISC-V 指令的超轻量级软核微控制器，可嵌入 FPGA，便于用软件实现一定复杂度的控制逻辑。目前 femto 在 MIT 协议下开源发布。femto 基本特性列表如下：

- 单处理器核
- 可定制的 IP 核
- 内部 32 位单总线
- 同步系统，单个时钟域
- 支持指令预取
- 小端字节序
- 二级流水线
- RV32EC 指令集，支持 Zfencei 扩展
- 多数指令为单周期指令
- 无中断机制，无高速缓存，不支持调试功能
- 内置 MCU 常用 IP 核，便于使用 SRAM/NOR/UART

# 模块说明

femto 的示意框图如下：



所有内部模块均在统一的时钟下工作，此时钟频率应由 femto.vh 中的 **SYSCLK\_FREQ** 反映。femto 实现时需人为确保 **SYSCLK\_FREQ** 正确。

下面会分块介绍 femto 的内部结构。

## 内部总线(Bus)

femto 采用的是“请求-应答”机制下的 32 位内部总线。总线请求只能由 Core 发起，总线应答也均由 Core 处理。总线信号及时序如下图：



### 提示：总线协议

**req** 和 **resp** 高电平时，每个时钟有效一次。**req** 由 Core 驱动，表示发出总线请求。**resp** 由 IP 核驱动，表示当次总线请求完成。在 **resp** 到来时，下一 **req** 同步发出，提升总线利用率。

**addr**, **w/r**, **acc** 和 **wdata** 为总线请求的控制信号和数据，均由 Core 驱动。

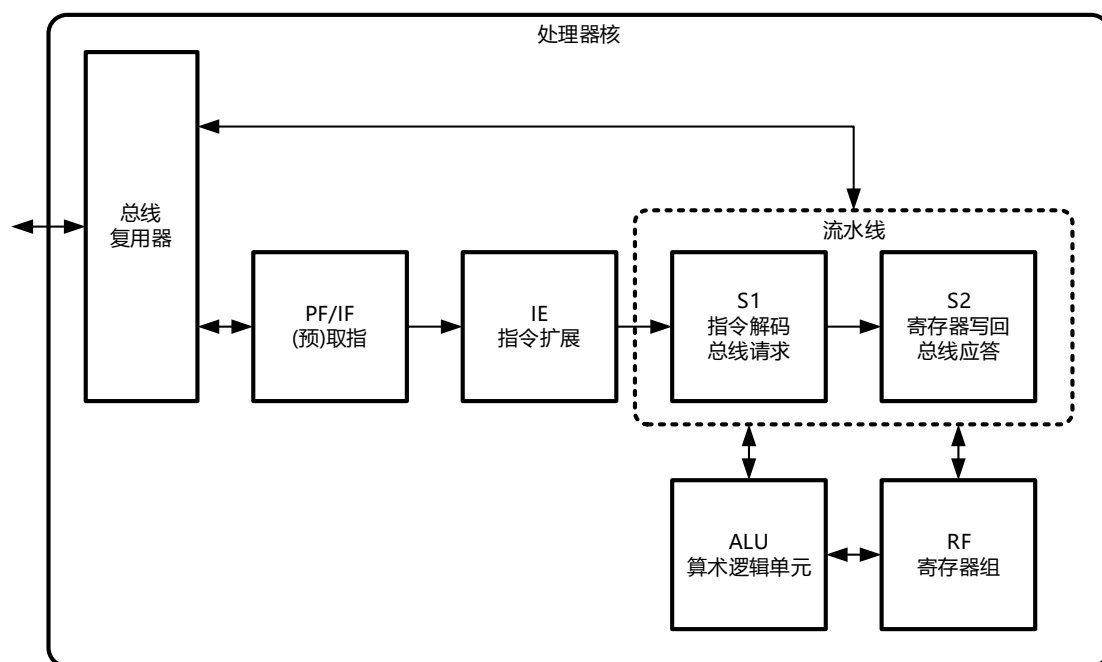
- 每个 IP 核独占一个总线地址区间，总线请求只会被派送到与 **addr** 匹配的 IP 核。如果 **addr** 不对应任何 IP 核，会触发总线错误。femto 最大支持 4GB 地址空间，原则上 femto 总线仅支持对齐访问，例如发起 32 位总线请求时，**addr** 应 4 字节对齐。
- w/r** 高电平表示当前总线请求为写请求，反之为读请求。
- acc** 用于指定总线请求位宽，支持 8 位(单字节)请求，16 位(双字节)请求以及 32 位(四字节)请求。
- wdata** 为写请求 32 位数据，总线写请求时 IP 核需处理 **wdata**，读请求时 IP 核不处理。对于 8 位写请求，Core 只保证 **wdata** 的低 8 位有效，IP 也只应处理低 8 位，依此类推。

**rdata** 是读总线请求的 32 位应答数据，由响应请求的 IP 核驱动。

- 读请求应答时 **rdata** 同时有效，对于 8 位请求，IP 只需保证 **rdata** 的低 8 位有效，同时 Core 也只需处理低 8 位，依此类推。

## 处理器核(Core)

处理器核的结构框图如下：



femto Core 分时复用总线，完成指令请求(取指)和数据请求(读写)，即采用了类似冯-诺伊曼/普林斯顿结构的总线形式。

## 内部只读存储器(ROM)

Core 可以经由内部总线直接读取 ROM。ROM 与 Core/Bus 运行在同一频率下，因此对 ROM 的访问仅需一个时钟即可完成。

此 IP 核无寄存器，总线地址由 femto.vh 中的 **ROM\_ADDR** 指定，典型值 0x00000000。

ROM 一般用于存储系统引导程序(Bootloader)，ROM 的内容在 femto 实现时已经由 rom.vh 指定。

## 内部密耦合存储器(TCM)

Core 可以经由内部总线直接读写 TCM。TCM 与 Core/Bus 运行在同一频率下，因此对 TCM 的访问仅需一个时钟即可完成。

此 IP 核无寄存器，总线地址由 femto.vh 中的 **TCM\_ADDR** 指定，典型值 0x10000000。

## 外部存储(SRAM)控制器

SRAM 控制器支持访问外部异步 SRAM。通过此控制器，Core 可以经由内部总线读写外部 SRAM。

此 IP 核无寄存器，总线地址由 femto.vh 中的 **SRAM\_ADDR** 指定，典型值 0x20000000。

## 外部闪存(NOR)读控制器

NOR 读控制器支持常见的 Quad SPI Serial NOR 产品。此控制器将内部读外部 NOR 的总线请求映射到片选，串行时钟和 4 位数据信号上。

此 IP 核无寄存器，总线地址由 femto.vh 中的 **NOR\_ADDR** 指定，典型值 0x30000000。

此控制器与后文的 QSPINOR 控制器实质上为同一个模块，需要正确配置 QSPINOR 控制器内的相应寄存器才可使用 NOR 读控制器。

# 通用输入输出引脚(GPIO)控制器

GPIO 控制器可以驱动或读取 femto 的对外引脚,可以控制最多 32 个引脚。实际控制的引脚数由 femto.vh 中的 **GPIO\_WIDTH** 指定, 典型值 4。

此 IP 核需通过寄存器使用, 寄存器总线地址由 femto.vh 中的 **GPIO\_ADDR** 指定, 典型值 0x40000000。  
寄存器定义如下:

| 寄存器 | 偏移地址 | 位宽 | 访问权限 | 注释          |
|-----|------|----|------|-------------|
| D   | 0x0  | 32 | 读写   | 引脚电平(读取/设置) |
| DIR | 0x4  | 32 | 读写   | 引脚方向(输入/输出) |

● D

| 位   | 31                                      | ... | 0                                     |
|-----|---|-----|---------------------------------------|
| 读功能 | 输入状态:读取引脚 31 电平<br>输出状态:读取引脚 31 输出电平设置值 | ... | 输入状态:读取引脚 0 电平<br>输出状态:读取引脚 0 输出电平设置值 |
| 写功能 | 输入状态:预设引脚 31 输出电平<br>输出状态:设置引脚 31 输出电平  | ... | 输入状态:预设引脚 0 输出电平<br>输出状态:设置引脚 0 输出电平  |

● DIR

| 位   | 31            | ... | 0            |
|-----|---------------|-----|--------------|
| 读功能 | 0:引脚 31 为输入引脚 | ... | 0:引脚 0 为输入引脚 |
| 写功能 | 1:引脚 31 为输出引脚 | ... | 1:引脚 0 为输出引脚 |

# 异步串口(UART)控制器

UART 控制器采用固定波特率。其波特率由 femto.vh 中的 **UART\_BAUD** 指定, 典型值 57600。UART 控制器中有发送/接收 FIFO, 其深度均由 femto.vh 中的 **UART\_FIFO\_DEPTH** 指定, 典型值 8。当发送 FIFO 中有数据时, 此 IP 核会依次发出这些数据。当此 IP 核接收到数据时, 也会依次存入接收 FIFO。

此 IP 核需通过寄存器使用, 寄存器总线地址由 femto.vh 中的 **UART\_ADDR** 指定, 典型值 0x50000000。  
寄存器定义如下:

| 寄存器    | 偏移地址 | 位宽 | 访问权限 | 注释            |
|--------|------|----|------|---------------|
| TXD    | 0x0  | 8  | 只写   | 发送数据          |
| RXD    | 0x1  | 8  | 只读   | 接收数据          |
| TXQSR  | 0x2  | 8  | 只读   | 发送 FIFO 状态    |
| RXQCSR | 0x3  | 8  | 读写   | 接收 FIFO 控制与状态 |

● TXD

| 位   | 7~0                           |
|-----|-------------------------------|
| 读功能 | N/A                           |
| 写功能 | 待发送字节。若发送 FIFO 已满, 写此寄存器会被忽略。 |

● RXD

| 位   | 7~0                          |
|-----|------------------------------|
| 读功能 | 已接收字节。若接收 FIFO 为空, 此寄存器值未定义。 |

|     |     |
|-----|-----|
| 写功能 | N/A |
|-----|-----|

#### ● TXQSR

|     |     |                                     |
|-----|-----|-------------------------------------|
| 位   | 7~1 | 0                                   |
| 读功能 | N/A | 0:发送 FIFO 已满/不可写<br>1:发送 FIFO 未滿/可写 |
| 写功能 |     | N/A                                 |

#### ● RXQCSR

|     |     |                      |                                     |
|-----|-----|----------------------|-------------------------------------|
| 位   | 7~2 | 1                    | 0                                   |
| 读功能 | N/A | N/A                  | 0:接收 FIFO 为空/不可读<br>1:接收 FIFO 非空/可读 |
| 写功能 |     | 1:清空接收 FIFO<br>0:无作用 | N/A                                 |

## 四线同步闪存(QSPINOR)控制器

QSPINOR 控制器与 NOR 控制器实质上是同一个 IP 核，但是与 NOR 读控制器“总线直接读取 NOR”不同，QSPINOR 通过寄存器实现对 NOR 的访问。另外 QSPINOR 控制器可以实现灵活的读/写/擦等一般访问，而 NOR 读控制器只允许读访问。

### 警告：NOR XiP 程序不应访问 QSPINOR 控制器

NOR 读控制器会等待 QSPINOR 完成活动(片选信号关停)，期间 NOR 会挂起总线。因此在 QSPINOR 片选有效期间，总线上不应有对 NOR 的访问。尤其要指出 NOR XiP 程序不应访问 QSPINOR 控制器，否则会导致总线死锁。

QSPINOR 控制器内部有收/发 FIFO 用来暂存已收/待发数据。FIFO 深度由 femto.vh 中的 **QSPINOR\_FIFO\_DEPTH** 指定，典型值为 16。

此 IP 核的寄存器总线地址由 femto.vh 中的 **QSPINOR\_ADDR** 指定，典型值 0x60000000。

寄存器定义如下：

| 寄存器    | 偏移地址 | 位宽 | 访问权限 | 注释            |
|--------|------|----|------|---------------|
| IPCSR  | 0x0  | 16 | 读写   | NOR 访问命令      |
| DATA   | 0x2  | 8  | 读写   | 收发数据          |
| TXQCSR | 0x3  | 8  | 读写   | 接收 FIFO 控制与状态 |
| RXQCSR | 0x4  | 8  | 读写   | 发送 FIFO 控制与状态 |
| NORCMD | 0x5  | 8  | 读写   | NOR 读控制器读命令字  |
| NORCSR | 0x6  | 16 | 读写   | NOR 读控制器控制与状态 |

#### ● IPCSR

写 IPCSR 会触发 QSPINOR 控制器与外部 NOR 交互。本小节视这一过程为一次“IPCSR 命令执行”。

|     |  |          |                                      |                          |
|-----|--|----------|--------------------------------------|--------------------------|
| 位   | 15~12                                    | 11~8     | 7~6                                  | 5                        |
| 读功能 | N/A                                      | N/A      | N/A                                  | N/A                      |
| 写功能 | 输出信号值，仅在输出占位周期时有效<br><br>12 位:数据线[0]输出电平 | 命令动作重复次数 | 0:单线模式<br>1:双线模式<br>2:四线模式<br>其它:未定义 | 0:此命令为数据交互<br>1:此命令为占位周期 |

|     |                        |     |                         |                              |
|-----|------------------------|-----|-------------------------|------------------------------|
|     | ...<br>15 位:数据线[3]输出电平 |     |                         |                              |
| 位   | 4                      | 3~2 | 1                       | 0                            |
| 读功能 | N/A                    | N/A | 0:无命令正在执行<br>1:前一命令正在执行 | 0:命令结束, 片选关停<br>1:命令执行, 片选有效 |
| 写功能 | 0:输入<br>1:输出           |     | N/A                     |                              |

此寄存器的定义较为复杂, 这里给出一些命令示例:

|                             |  |
|-----------------------------|--|
| 单线发出 1 字节(可用于发出 NOR 指令字)    | 写值 0x0111                                |
| 双线发出 3 字节(可用于发出 NOR 地址)     | 写值 0x0351                                |
| 四线连续读取, 直到接收 FIFO 满         | 写值 0x0081                                |
| 四线连续发送, 直到发送 FIFO 空         | 写值 0x0091                                |
| 双线读取 4 字节                   | 写值 0x0441                                |
| 4 个占位周期, 期间数据线对外呈输入态        | 写值 0x0421                                |
| 16 个占位周期, 期间数据线对外呈输入态       | 写值 0x0021                                |
| 1 个占位周期, 期间数据线[1:0]对外输出 b10 | 写值 0x2171                                |
| 关停片选信号                      | 写值 0x0000                                |
| 查看命令执行状态                    | 读 IPCSR, 第 0 位表示片选有效与否, 第 1 位表示是否有命令正在执行 |

大多数对 QSPINOR 控制器的总线请求会在一个时钟周期后得到应答, 只有一个例外: 连续写 IPCSR 命令时, 如果前一命令尚未完成, QSPINOR 控制器会保存第二次 IPCSR 并挂起总线, 待到前一命令执行完成, 放开总线并自动续接执行第二条命令。这种方式被视为“阻塞模式”。

如需避免总线挂起, 应在写 IPCSR 之前读取并确认 IPCSR[1]为 0。这种方式被视为“非阻塞模式”。在非阻塞模式下, 若 IPCSR[11:8]为 0, 读(写)命令会持续到接收(发送)FIFO 满(空); 若在下达命令时, FIFO 已经满(空), Core 应当读取(写入)FIFO, 而后命令会执行到 FIFO 再次满(空)。若 IPCSR[11:8]非 0, 读(写)命令会确保执行 IPCSR[11:8]次; 在命令执行期间, 若 FIFO 满(空), 则命令暂停, 需要 Core 读取(写入)FIFO 以继续。

通过组合多个命令, 可以实现一个完整的 NOR 访问指令。如“0x0111 0x0351 0x0141 0x0000”命令序列可以实现典型的 SPI NOR 单线读字节指令。相邻命令之间往往会出现空闲间隙, 间隙中串行 SPI 时钟暂停。

#### 提示: 连续 NOR 访问

灵活使用 QSPINOR 控制器可以实现对 NOR 的连续访问, 从而提升大数据量访问的效率。在非阻塞模式下, IPCSR[11:8]为 0 时, 读(写)命令会持续到接收(发送)FIFO 满(空)。利用这一特性, Core 可在发起连续读(写)后持续读取(写入)FIFO, 从而实现大量数据的喷发式接收(发送)。

原则上 QSPINOR 控制器支持这种用法, 但是尚未就此专门优化。如果读写 NOR 的速度较快, 而 Core 持续访问 FIFO 时平均速度较慢, 可能迫使喷发式数据传输中止。

#### ● DATA

|     |                               |
|-----|-------------------------------|
| 位   | 7~0                           |
| 读功能 | 已接收字节。若接收 FIFO 为空, 此寄存器值未定义。  |
| 写功能 | 待发送字节。若发送 FIFO 已满, 写此寄存器会被忽略。 |

#### ● TXQCSR

| 位   | 7                          | 6:5 | 4:0            |
|-----|----------------------------|-----|----------------|
| 读功能 | N/A                        | N/A | 发送 FIFO 中的空单元数 |
| 写功能 | 1:无命令执行时清空发送 FIFO<br>0:无作用 |     | N/A            |

● RXQCSR

| 位   | 7                    | 6:5 | 4:0             |
|-----|----------------------|-----|-----------------|
| 读功能 | N/A                  | N/A | 接收 FIFO 中的有效数据数 |
| 写功能 | 1:清空接收 FIFO<br>0:无作用 |     | N/A             |

● NORCMD

| 位   | 7~0         |
|-----|-------------|
| 读功能 | NOR 读控制器读命令 |
| 写功能 |             |

● NORCSR

此寄存器用于配置 NOR 读控制器的工作模式。只有正确配置此寄存器 NOR 读控制器才能正常工作。

| 位   | 15~12 | 11~8                                      | 7~4   | 3                                    | 2~0  |
|-----|-------|---|-------|--------------------------------------|--|
| 读功能 | N/A   | 输出信号值，仅在输出占位周期时有效                         | 占位周期数 | 0:占位周期期间串行数据为输入态<br>1:占位周期期间串行数据为输出态 | 0:1-1-1 模式   |
| 写功能 |       | 12 位:数据线[0]输出电平<br>...<br>15 位:数据线[3]输出电平 |       |                                      | 1:1-1-2 模式<br>2:1-1-4 模式<br>3:1-2-2 模式<br>4:1-4-4 模式<br>5:2-2-2 模式<br>6:4-4-4 模式<br>其它:未定义 |

## 计时器

计时器可以为软件提供较为准确的计时功能。计时时钟由全局时钟分频得到，分频倍率为由 femto.vh 中的 **TMR\_DIV** 指定，故而计时时钟频率为 **SYSCLK\_FREQ/TMR\_DIV**。**TMR\_DIV** 的典型值是 24。

此 IP 核需通过寄存器使用，寄存器总线地址由 femto.vh 中的 **TMR\_ADDR** 指定，典型值 0x70000000。

寄存器定义如下：

| 寄存器 | 偏移地址 | 位宽 | 访问权限 | 注释     |
|-----|------|----|------|--------|
| TR  | 0x0  | 32 | 读写   | 计时器计数值 |

● TR

| 位   | 31~0     |
|-----|----------|
| 读功能 | 计时器实时计数值 |
| 写功能 |          |

TR 寄存器非 0 时，每个计时时钟自减 1，直至自减到 0 为止。读取 TR 可以获得当前计数值。TR 寄存器随时可写。



## 复位控制器

复位控制器提供 femto 内部各个时序逻辑的复位功能。外部输入复位信号(低电平有效)可以触发全局复位，Core 也可通过此 IP 核实现软件复位。

此 IP 核需要通过寄存器使用，寄存器总线地址由 femto.vh 中的 **RST\_ADDR** 定义，典型值 0xf0000000。寄存器定义如下：

| 寄存器 | 偏移地址 | 位宽 | 访问权限 | 注释   |
|-----|------|----|------|------|
| RST | 0x0  | 8  | 只写   | 复位控制 |

● RST

|     |   |
|-----|---|
| 位   | 7~0   |
| 读功能 | N/A   |
| 写功能 | 0:触发全局复位<br>1:复位目标模块 0<br>...<br>255:复位目标模块 254 |

目前各个复位目标模块在 femto.v 中指定，建议在 femto.vh 中记录各个目标模块。不建议复位程序/指令存储器。典型的目标模块定义如下：

| 复位目标 | 注释             |
|------|----------------|
| 0    | 复位处理器核心        |
| 1    | 复位 ROM 控制器     |
| 2    | 复位 TCM 控制器     |
| 3    | 复位 SRAM 控制器    |
| 4    | 复位 NOR 读控制器    |
| 5    | 复位 GPIO 控制器    |
| 6    | 复位 UART 控制器    |
| 7    | 复位 QSPINOR 控制器 |
| 8    | 复位计时器          |

## 输入输出环(I/O Ring)

I/O Ring 统一控制对 femto 外信号的输入/输出方向。

## 软件开发包(SDK)

软件开发包封装了对软件对 femto 寄存器的操作并抽象出常用的函数。SDK 可以简化、加速软件开发，避免或减少软件开发过程中的问题。SDK 功能通过 sdk.h, sdk.c 提供。  
下面分模块给出 SDK 的详细信息。

### NOR 读控制器

```
void nor_init(  
    nor_mode_t mode,
```

```
uint8_t cmd,
uint8_t dmy_cycle_no,
bool dmy_out,
uint8_t dmy_out_pattern
);
```

| 功能              | 配置 NOR 读控制器                      |
|-----------------|----------------------------------|
| mode            | NOR_MODE_111 ... NOR_MODE_444    |
| cmd             | NOR 读命令字                         |
| dmy_cycle_no    | 占位周期数, 0~15                      |
| dmy_out         | 0:占位周期期间数据线对外高阻, 1:占位周期期间数据线对外输出 |
| dmy_out_pattern | 占位周期期间数据线对外输出的电平                 |

## GPIO 控制器

```
gpio_dir_t gpio_get_dir(uint8_t pin_index);
```

| 功能        | 获取引脚输入/输出状态(方向) |
|-----------|-----------------|
| pin_index | 引脚下标, 0~31      |
| 返回值       | DIR_IN, DIR_OUT |

```
void gpio_set_dir(uint8_t pin_index, gpio_dir_t dir);
```

| 功能        | 设置引脚输入/输出状态(方向) |
|-----------|-----------------|
| pin_index | 引脚下标, 0~31      |
| dir       | DIR_IN, DIR_OUT |

```
bool gpio_get(uint8_t pin_index);
```

| 功能        | 获取输入引脚电平     |
|-----------|--------------|
| pin_index | 引脚下标, 0~31   |
| 返回值       | 0:低电平, 1:高电平 |

```
void gpio_set(uint8_t pin_index, bool level);
```

| 功能        | 设置输出引脚电平     |
|-----------|--------------|
| pin_index | 引脚下标, 0~31   |
| level     | 0:低电平, 1:高电平 |

```
void gpio_tog(uint8_t pin_index);
```

| 功能        | 反转输出引脚电平   |
|-----------|------------|
| pin_index | 引脚下标, 0~31 |

## UART 控制器

```
bool uart_rx_ready(void);
```

| 功能  | 检查 UART 接收 FIFO 是否可读        |
|-----|-----------------------------|
| 返回值 | 0:FIFO 为空/不可读, 1:FIFO 非空/可读 |

**bool** uart\_tx\_ready(**void**);

|     |                             |
|-----|-----------------------------|
| 功能  | 检查 UART 发送 FIFO 是否可写        |
| 返回值 | 0:FIFO 已满/不可写, 1:FIFO 未满/可写 |

**void** uart\_clear\_rx\_fifo(**void**);

|    |                 |
|----|-----------------|
| 功能 | 清空 UART 接收 FIFO |
|----|-----------------|

**bool** uart\_read\_rxq(**uint8\_t\*** **const** ptr\_d);

|       |                       |
|-------|-----------------------|
| 功能    | 尝试读取 UART 接收 FIFO 并存储 |
| ptr_d | 出参, 存储到此地址/指针         |
| 返回值   | 0:失败, 1: 成功           |

**bool** uart\_write\_txq(**uint8\_t** d);

|     |                  |
|-----|------------------|
| 功能  | 尝试写 UART 发送 FIFO |
| d   | 待发送字节            |
| 返回值 | 0:失败, 1: 成功      |

**void** uart\_receive\_data(**uint8\_t\*** **const** buf, **size\_t** n);

|     |                              |
|-----|------------------------------|
| 功能  | 从 UART 接收 FIFO 读取数据, 读取完成后返回 |
| buf | 接收目的地址/指针                    |
| n   | 待接收字节数                       |

**void** uart\_send\_data(**const** **uint8\_t\*** **const** buf, **size\_t** n);

|     |                              |
|-----|------------------------------|
| 功能  | 将数据串写入 UART 发送 FIFO, 写入完成后返回 |
| buf | 待发送字节串地址/指针                  |
| n   | 待发送字节数                       |

## QSPINOR 控制器

**bool** qspinor\_rxq\_ready(**void**);

|     |                             |
|-----|-----------------------------|
| 功能  | 检查 QSPINOR 接收 FIFO 是否可读     |
| 返回值 | 0:FIFO 为空/不可读, 1:FIFO 非空/可读 |

**bool** qspinor\_txq\_ready(**void**);

|     |                             |
|-----|-----------------------------|
| 功能  | 检查 QSPINOR 发送 FIFO 是否可写     |
| 返回值 | 0:FIFO 已满/不可写, 1:FIFO 未满/可写 |

**void** qspinor\_clear\_rxq(**void**);

|    |                    |
|----|--------------------|
| 功能 | 清空 QSPINOR 接收 FIFO |
|----|--------------------|

**void** qspinor\_clear\_txq(**void**);

|    |                              |
|----|------------------------------|
| 功能 | 清空 QSPINOR 发送 FIFO(仅在未执行命令时) |
|----|------------------------------|

```
bool qspinor_read_rxq(uint8_t* const ptr_d);
```

|       |                          |
|-------|--------------------------|
| 功能    | 尝试读取 QSPINOR 接收 FIFO 并存储 |
| ptr_d | 出参, 存储到此地址/指针            |
| 返回值   | 0:失败, 1: 成功              |

```
bool qspinor_write_txq(uint8_t d);
```

|     |                     |
|-----|---------------------|
| 功能  | 尝试写 QSPINOR 发送 FIFO |
| d   | 待发送字节               |
| 返回值 | 0:失败, 1: 成功         |

```
bool qspinor_is_busy(void);
```

|     |                      |
|-----|----------------------|
| 功能  | 检查 QSPINOR 当前是否在执行命令 |
| 返回值 | 0:未在执行命令, 1:正在执行命令   |

```
void qspinor_begin_receive(uint8_t n, qspinor_width_t width);
```

|       |  |
|-------|--|
| 功能    | 发起一轮数据接收, 数据写入接收 FIFO, 尽快返回                        |
| n     | WHOLE_FIFO:触发一轮接收, 直至接收 FIFO 已满<br>1:接收一个字节<br>... |
| width | QSPINOR_X1, QSPINOR_X2, QSPINOR_X4                 |

```
void qspinor_begin_send(uint8_t n, qspinor_width_t width);
```

|       |  |
|-------|--|
| 功能    | 发起一轮数据发送, 数据源自发送 FIFO, 尽快返回                        |
| n     | WHOLE_FIFO:触发一轮发送, 直至发送 FIFO 为空<br>1:发送一个字节<br>... |
| width | QSPINOR_X1, QSPINOR_X2, QSPINOR_X4                 |

```
void qspinor_send_dummy_cycle(
    uint8_t n,
    qspinor_width_t width,
    bool dmy_out,
    uint8_t dmy_out_pattern
);
```

|                 |                                    |
|-----------------|------------------------------------|
| 功能              | 发起一轮占位周期, 尽快返回                     |
| n               | 0(16):16 个占位周期<br>1:一个占位周期<br>...  |
| width           | QSPINOR_X1, QSPINOR_X2, QSPINOR_X4 |
| dmy_out         | 0:占位周期期间数据线对外高阻, 1:占位周期期间数据线对外输出   |
| dmy_out_pattern | 占位周期期间数据线对外输出的电平                   |

```
void qspinor_finish(void);
```

|    |                                  |
|----|----------------------------------|
| 功能 | 结束 QSPINOR 与 NOR 的交互, 关停片选, 尽快返回 |
|----|----------------------------------|

```
void qspinor_receive_data(
    uint8_t* const buf,
    size_t n,
    qspinor_width_t width
);
```

|       |                                    |
|-------|------------------------------------|
| 功能    | 用 QSPINOR 接收指定长度数据，接收完成后返回         |
| buf   | 接收目的地址/指针                          |
| n     | 待接收字节数                             |
| width | QSPINOR_X1, QSPINOR_X2, QSPINOR_X4 |

```
void qspinor_send_data(
    const uint8_t* const buf,
    size_t n,
    qspinor_width_t width
);
```

|       |                                    |
|-------|------------------------------------|
| 功能    | 用 QSPINOR 发送指定长度数据，发送完成后返回         |
| buf   | 待发送字节串地址/指针                        |
| n     | 待发送字节数                             |
| width | QSPINOR_X1, QSPINOR_X2, QSPINOR_X4 |

## 计时器

```
void timer_set(uint32_t val);
```

|     |   |
|-----|---|
| 功能  | 写入计时器计数初值，立即返回                                    |
| val | 0:清空计数值并停止计数<br>正数:设置计数初值，此后每个计时时钟计数值减 1，直至计数值为 0 |

```
uint32_t timer_get(void);
```

|     |              |
|-----|--------------|
| 功能  | 读取计时器当前实时计数值 |
| 返回值 | 实时计数值        |

```
void timer_delay_us(uint32_t val);
```

|     |             |
|-----|-------------|
| 功能  | 阻塞式延时，完成后返回 |
| val | 延时长度（微秒）    |

## 复位控制器

```
void reset_soc(void);
```

|    |          |
|----|----------|
| 功能 | 复位 femto |
|----|----------|

```
void reset_core(void);
```

|    |              |
|----|--------------|
| 功能 | 停止处理器核的活动并复位 |
|----|--------------|

```
void reset_uart(void);
```

|    |               |
|----|---------------|
| 功能 | 停止串口控制器的活动并复位 |
|----|---------------|

```
void reset_gpio(void);
```

|    |                   |
|----|-------------------|
| 功能 | 停止 GPIO 控制器的活动并复位 |
|----|-------------------|

```
void reset_qspinor(void);
```

|    |                      |
|----|----------------------|
| 功能 | 停止 QSPINOR 控制器的活动并复位 |
|----|----------------------|

```
void reset_timer(void);
```

|    |             |
|----|-------------|
| 功能 | 停止计时器的活动并复位 |
|----|-------------|

## 开发与运行环境

目前 femto 的开发主要需要两款工具软件：

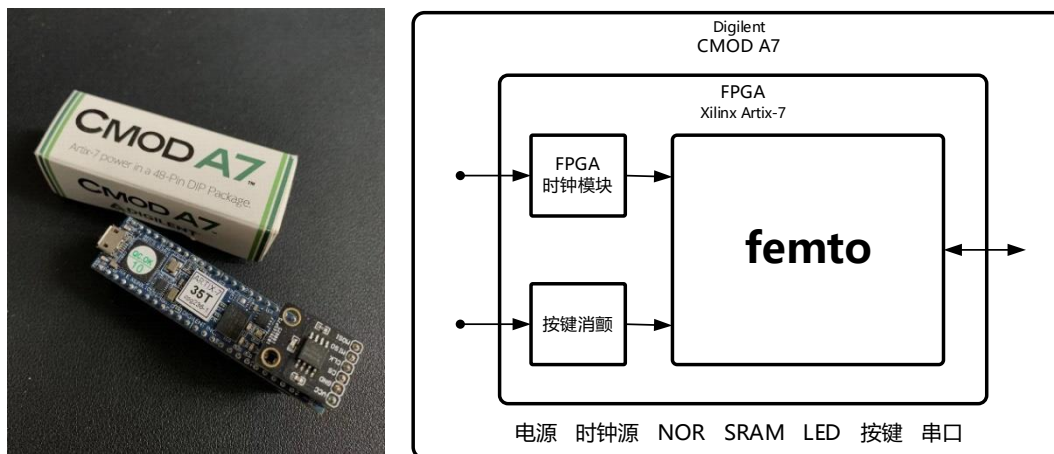
gcc version 8.3.0 (xPack GNU RISC-V Embedded GCC, 32-bit)

Xilinx Vivado v2018.2 (64-bit)

SW Build: 2258646 on Thu Jun 14 20:03:12 MDT 2018

IP Build: 2256618 on Thu Jun 14 22:10:49 MDT 2018

femto 可以在实体硬件上运行，其典型运行平台如下图所示：



## 板级软件开发包(BSDK)

板级软件开发包可以进一步简化在典型平台上的软件开发。BSDK 功能通过相应的头文件和 C 文件提供。目前 BSDK 主要涵盖了对 GPIO、NOR 闪存的支持，其主要内容如下：

**警告：NOR XiP 程序不应调用 QSPINOR 相关的函数**

这可能会导致总线死锁。详细信息参考 QSPINOR 控制器介绍章节。

```
void nor_bus_read_init(void);
```

|    |                     |
|----|---------------------|
| 功能 | 使用典型配置初始化 NOR 读控制模块 |
|----|---------------------|

```
void nor_erase_block(size_t block_offset);
```

|    |                       |
|----|-----------------------|
| 功能 | 擦除 NOR 闪存指定 Block 的数据 |
|----|-----------------------|

```
void nor_program(  
    size_t start_page_offset,  
    const uint8_t* const data,  
    size_t size  
);
```

|    |                      |
|----|----------------------|
| 功能 | 向 NOR 闪存已擦除的指定区间写入数据 |
|----|----------------------|

```
void nor_read(size_t byte_offset, uint8_t* const data, size_t size);
```

|    |   |
|----|---|
| 功能 | 通过 QSPINOR 命令读取 NOR 闪存指定区间的数据<br>(并非通过总线读取) |
|----|---|

```
void gpio_init(void);
```

|    |                   |
|----|-------------------|
| 功能 | 将 GPIO 初始化至板卡默认状态 |
|----|-------------------|

```
bool get_button_level(void);
```

|    |           |
|----|-----------|
| 功能 | 获取用户按键的电平 |
|----|-----------|

```
void light_leds(bool red, bool green, bool blue);
```

|    |              |
|----|--------------|
| 功能 | 点亮板载三色 LED 灯 |
|----|--------------|

# 系统引导(Boot)

用户可以通过 femto 内部的超轻量级引导程序 (Boot loader), 加载并执行自己开发的应用程序 (App)。femto 支持从串口引导或从 NOR 闪存引导。  
引导程序会在 femto 的片内 ROM 中执行, 并会使用片内 TCM 作为运行内存。

## 串口引导-SRAM 执行

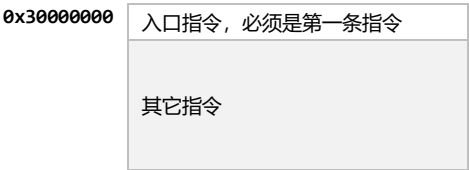
在 femto 运行的典型硬件平台 (CMOD A7) 上, 引导程序启动时会点亮板载的绿色 LED 并呼吸式闪烁。在此期间, 用户可通过串口向引导程序发送 App 可执行文件。引导程序将文件流式写入 SRAM 的起始地址。串口传输有断流超时值, 典型超时值为 0.5 秒。引导程序将断流视为传输完成。传输完成后, 引导程序会跳转到 SRAM 的起始地址, 开始执行, 完成引导。  
串口引导的 App 必须在 SRAM 的起始地址空间执行, 且入口指令必须是第一条指令, App 地址空间如下图所示:



## 闪存引导-XiP 执行

在 CMOD A7 上, 引导程序驱动板载绿色 LED 呼吸闪烁若干次后, 若无串口活动, 引导程序会跳转到 NOR 闪存的起始地址, 开始执行, 完成引导。  
与串口引导的 App 类似, 闪存引导的 App 必须在闪存的起始地址空间 XiP 执行, 且入口指令必须是第一

条指令，如下图所示：



**注意：App 类型**

femto 支持的 App 分为 SRAM 执行和 NOR 执行两类，可以将两类都视为 “XiP(eXecute-in-Place)” 的 App。由于 femto 引导程序直接运行 “裸可执行文件”，其中不含其它信息，故不支持非 XiP 的 App。比如某 App 设计运行于 SRAM，但希望存储于 NOR 中，闪存引导并不会加载此 App，相反可能触发错误。

闪存引导的 App 可执行文件需要事先写入 NOR 闪存中。为了使用便捷以及演示 SDK 功能，femto 提供了可由串口引导的闪存写入工具（Flash loader）。

## 闪存写入工具(Flash loader)的使用

闪存写入工具实质上是串口引导、SRAM 执行的 App。编译 Flash loader，生成二进制文件（典型文件名为 out-fld.bin），将其通过串口发送给 femto 引导程序。串口预设模式如下：

| 调制速率       | 校验位 | 数据  | 停止位   |
|------------|-----|-----|-------|
| 57600 Baud | 无   | 8 位 | 有，1 位 |

在 CMOD A7 上 Flash loader 会点亮板载的蓝色 LED 并呼吸式闪烁。在此期间，用户可通过串口向 Flash loader 发送用户 App 的可执行文件。femto 提供了用于测试的示例 App，可编译生成二进制文件（典型文件名为 out-app.bin）。

将 App 可执行文件通过串口发送，Flash loader 会将其缓存在 SRAM 中。串口传输有断流超时值，典型超时值为 0.5 秒。Flash loader 将断流视为传输完成。传输完成后，Flash loader 将缓存的 App 可执行文件写入闪存的起始地址。写入完成后，Flash loader 会跳转到闪存起始地址，开始执行，实现 “二级引导” 功能。

**注意：App 类型**

Flash loader 的工作形式与 Boot loader 的串口引导较为相似，但是注意 Flash loader 只应接受闪存引导的 App。