STM32F042K6 Datasheet: https://www.st.com/resource/en/datasheet/stm32f042c4.pdf

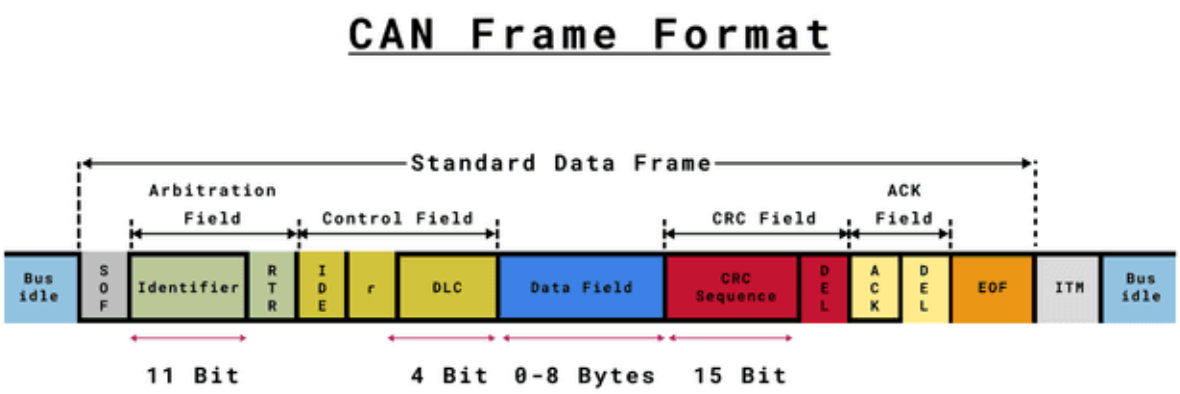
STM32 Programming Basics:

Starting with STM32 - Programming Tutorial for Beginners | Step by Step | Greidi Ajalik

Part 1: Basics of CAN Protocol

Basic introduction on CAN protocol: CAN Bus: Serial Communication - How It Works?

STM32 CAN LOOPBACK Mode || FILTER Configuration



RTR: Remote Transmission Request

- Defines if we are sending a data frame or a remote frame
- Dominant RTR bit (logic 0) indicates a data frame (used for sending actual data)
- Recessive RTR bit (logic 1) indicates a remote frame (used for requesting data from another node without including a data field itself)

IDE: Identifier Extension

- Determines whether the frame is using a standard or extended bit format
- Standard: 11-bit format
- Extended: 29-bit format
- IDE bit with a dominant (logic 0) bit indicates the frame uses the standard 11-bit identifier
- IDE bit with recessive (logic 1) bit indicates the frame uses the extended 29-bit identifier

R: Reserved Bit

- Varies depending on the protocol version
- Allows for future protocol extensions and maintains backward compatibility

DLC: Data Length Code

- The data length in bytes

- 4-bit field that specifies the number of bytes in the data field of the message, ranging from 0 to 8 bytes in standard CAN
- EX: A DLC of 1010 signifies there are 5 data bytes in the frame

Data Field

- The part of the frame that carries the actual payload or user data, containing between 0 and 8 bytes of information

STOPPED AT Time 1:45

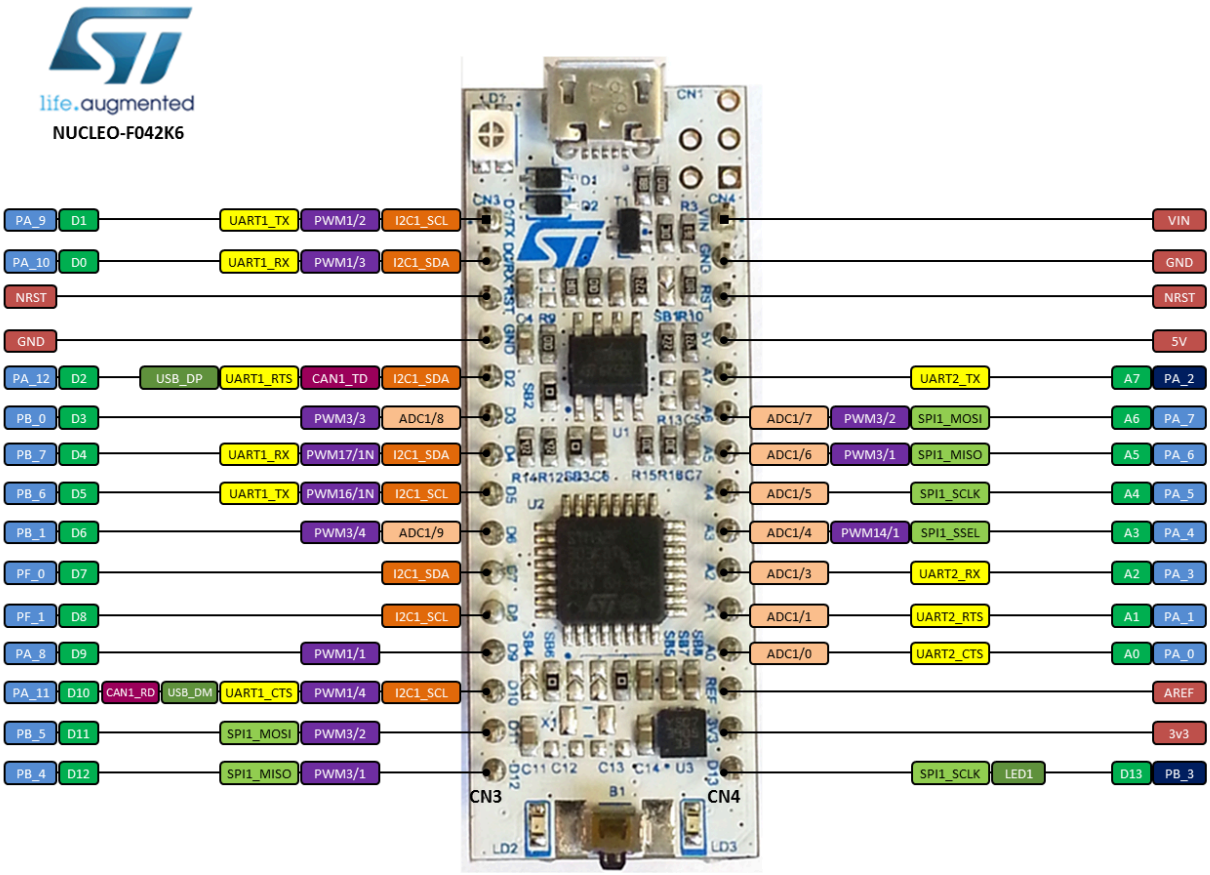
Part 2:

STM32 CAN Communication Explained | Send & Receive Data Between Two Microcontro…
Written tutorial

Maybe try this?

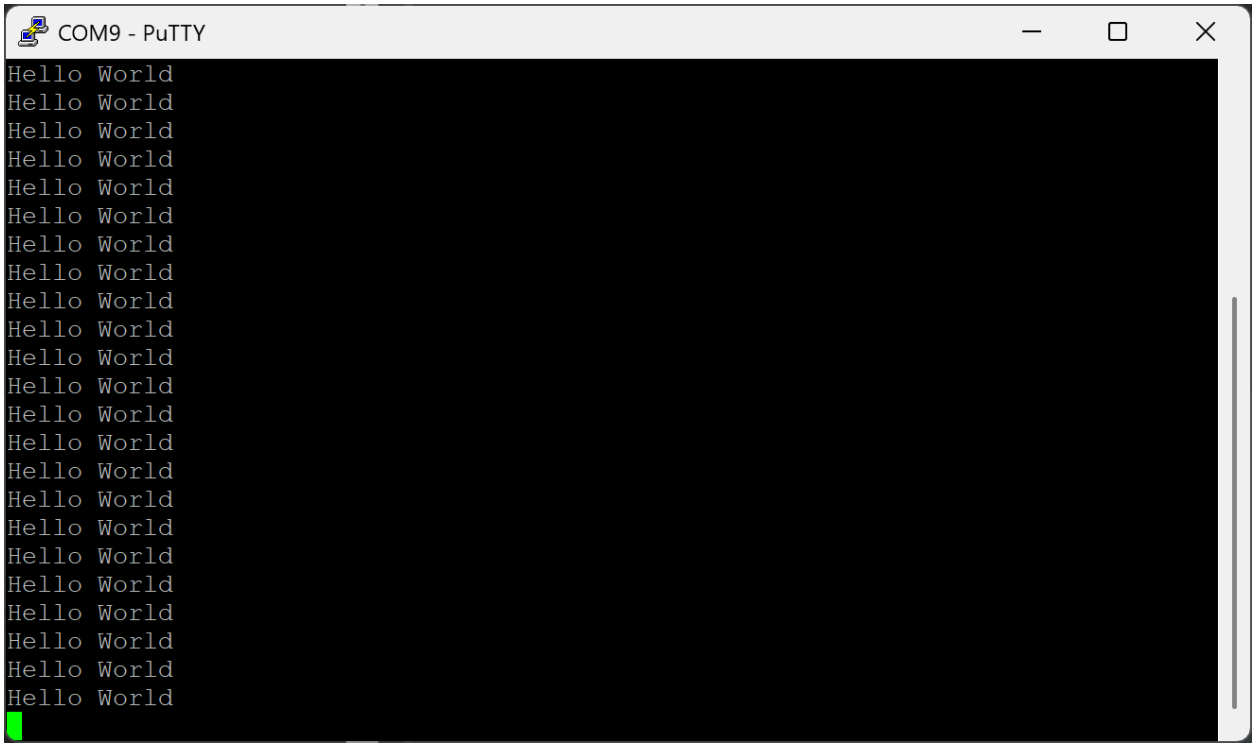
Additional Understanding of CAN Interface:

STM32 CAN Interface : 7 Steps - Instructables



Haniel's Programing Progress:

I was able to print Hello World to the PUTTY serial monitor.

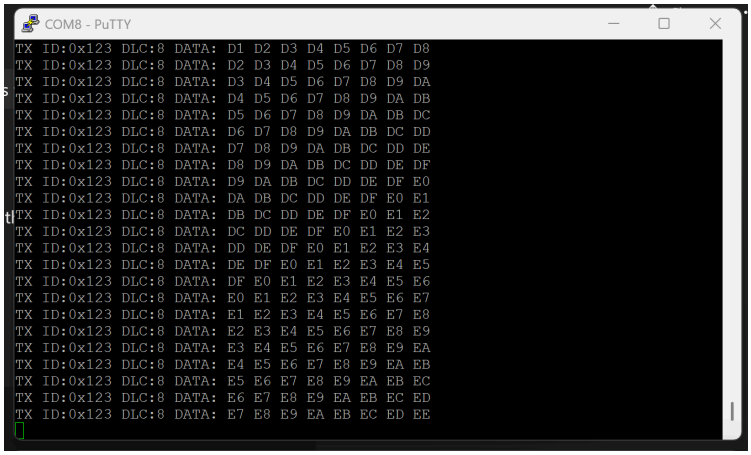


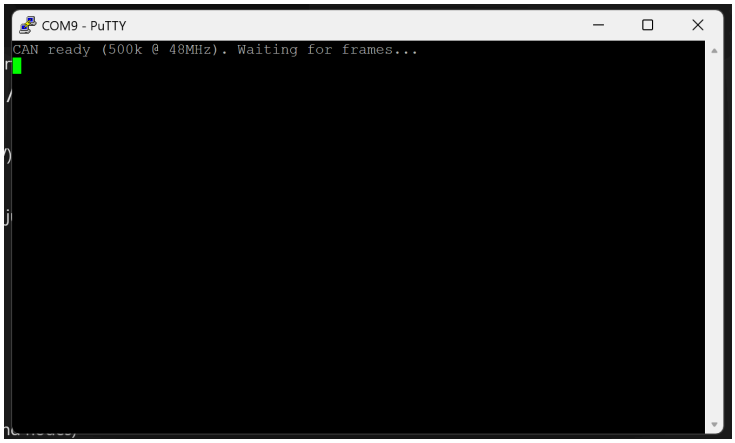
Here is the code to achieve this

```c
#include "main.h"
#include "stm32f0xx_hal.h"
#include <string.h>
UART_HandleTypeDef huart2;
/* Function prototypes */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 char msg[] = "Hello World\r\n";
 while (1)
 {
   HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
   HAL_Delay(1000);  // 1 second delay
 }
}
/* USART2 init function */
static void MX_USART2_UART_Init(void)
{
 huart2.Instance = USART2;
 huart2.Init.BaudRate = 9600;       // Match this in PuTTY
 huart2.Init.WordLength = UART_WORDLENGTH_8B;
 huart2.Init.StopBits = UART_STOPBITS_1;
 huart2.Init.Parity = UART_PARITY_NONE;
 huart2.Init.Mode = UART_MODE_TX_RX;
 huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart2) != HAL_OK)
 {
```

```
  // Initialization error
  while (1);
 }
}
/* GPIO init (not really used, but CubeMX generates it) */
static void MX_GPIO_Init(void)
{
 __HAL_RCC_GPIOA_CLK_ENABLE();
}
/* System Clock Configuration (auto-generated by CubeMX normally) */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 HAL_RCC_OscConfig(&RCC_OscInitStruct);
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                     |RCC_CLOCKTYPE_PCLK1;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0);
}
```

I was also able to get the Sender to Send, but the receiver gets nothing (might be due to 120ohm resistor, or lack of)





Here is the Sender code

```c
#include "main.h"
#include "stm32f0xx_hal.h"
#include <string.h>
#include <stdio.h>
#include <stdarg.h>
/* ---- Handles ---- */
UART_HandleTypeDef huart2;
CAN_HandleTypeDef  hcan;
/* ---- Prototypes ---- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_CAN_Init(void);
static void send_can_frame(uint32_t std_id, uint8_t *data, uint8_t dlc);
static void uprintf(const char *fmt, ...);
/* ====================== MAIN ====================== */
int main(void)
{
 HAL_Init();
 SystemClock_Config();     // 48 MHz from HSI48
 MX_GPIO_Init();
 MX_USART2_UART_Init();    // VCP → PuTTY
 MX_CAN_Init();            // CAN on PA11/PA12
 /* Filters aren't required to TRANSMIT, but Cube drivers want a filter configured if RX IRQs are used.
    We'll just configure "accept all" to keep things consistent. */
 CAN_FilterTypeDef filter = {0};
 filter.FilterBank = 0;
 filter.FilterMode = CAN_FILTERMODE_IDMASK;
 filter.FilterScale = CAN_FILTERSCALE_32BIT;
 filter.FilterIdHigh = 0x0000;
 filter.FilterIdLow  = 0x0000;
 filter.FilterMaskIdHigh = 0x0000;
 filter.FilterMaskIdLow  = 0x0000;
 filter.FilterFIFOAssignment = CAN_RX_FIFO0;
 filter.FilterActivation = ENABLE;
 HAL_CAN_ConfigFilter(&hcan, &filter);
 if (HAL_CAN_Start(&hcan) != HAL_OK) {
  uprintf("CAN start failed\r\n");
  Error_Handler();
 }
 uprintf("CAN sender ready (ID 0x123 @ 500k). Sending every 1s...\r\n");
 uint8_t cnt = 0;
 for (;;)
 {
  uint8_t payload[8] = {
   cnt, (uint8_t)(cnt+1), (uint8_t)(cnt+2), (uint8_t)(cnt+3),
   (uint8_t)(cnt+4), (uint8_t)(cnt+5), (uint8_t)(cnt+6), (uint8_t)(cnt+7)
  };
  send_can_frame(0x123, payload, 8);
  uprintf("TX ID:0x123 DLC:8 DATA:");
  for (int i = 0; i < 8; i++) uprintf(" %02X", payload[i]);
  uprintf("\r\n");
  cnt++;
  HAL_Delay(1000);
 }
}
/* ================== Helpers ================== */
static void send_can_frame(uint32_t std_id, uint8_t *data, uint8_t dlc)
{
 CAN_TxHeaderTypeDef txh = {0};
 uint32_t mailbox;
 txh.StdId = std_id;
 txh.ExtId = 0;
 txh.IDE   = CAN_ID_STD;
 txh.RTR   = CAN_RTR_DATA;
 txh.DLC   = dlc;
 txh.TransmitGlobalTime = DISABLE;
 /* Wait if all 3 mailboxes are busy (simple backoff) */
```

```
uint32_t t0 = HAL_GetTick();
while (HAL_CAN_GetTxMailboxesFreeLevel(&hcan) == 0U)
{
  if ((HAL_GetTick() - t0) > 10) break; // don't stall forever
}
if (HAL_CAN_AddTxMessage(&hcan, &txh, data, &mailbox) != HAL_OK) {
  uprintf("AddTxMessage failed (bus off or no mailbox)\r\n");
}
}
/* ===== UART printf ===== */
static void uprintf(const char *fmt, ...)
{
 char buf[128];
 va_list ap; va_start(ap, fmt);
 int n = vsnprintf(buf, sizeof(buf), fmt, ap);
 va_end(ap);
 if (n < 0) return;
 if (n > (int)sizeof(buf)) n = sizeof(buf);
 HAL_UART_Transmit(&huart2, (uint8_t*)buf, (uint16_t)n, HAL_MAX_DELAY);
}
/* ================== Peripheral inits ================= */
/* 48 MHz system clock using HSI48 (F0-safe) */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48;
 RCC_OscInitStruct.HSI48State     = RCC_HSI48_ON;
 RCC_OscInitStruct.PLL.PLLState   = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();
 RCC_ClkInitStruct.ClockType      = RCC_CLOCKTYPE_HCLK |
                                    RCC_CLOCKTYPE_SYSCLK |
                                    RCC_CLOCKTYPE_PCLK1;
 RCC_ClkInitStruct.SYSCLKSource   = RCC_SYSCLKSOURCE_HSI48;
 RCC_ClkInitStruct.AHBCLKDivider  = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK) Error_Handler();
}
/* GPIO (nothing special needed here) */
static void MX_GPIO_Init(void)
{
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
}
/* USART2 (VCP): PA2=TX (AF1), PA15=RX (AF1) on Nucleo-32 */
static void MX_USART2_UART_Init(void)
{
 huart2.Instance = USART2;
 huart2.Init.BaudRate   = 9600;
 huart2.Init.WordLength = UART_WORDLENGTH_8B;
 huart2.Init.StopBits   = UART_STOPBITS_1;
 huart2.Init.Parity     = UART_PARITY_NONE;
 huart2.Init.Mode       = UART_MODE_TX_RX;
 huart2.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart2) != HAL_OK) Error_Handler();
}
/* CAN on PA11(CAN_RX)/PA12(CAN_TX) @ 500 kbps (48 MHz clock) */
static void MX_CAN_Init(void)
{
 hcan.Instance = CAN;
 hcan.Init.Prescaler = 6;               // 48MHz / (6 * (1+13+2)) = 500 kbps
 hcan.Init.Mode = CAN_MODE_NORMAL;
 hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
 hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
 hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
 hcan.Init.TimeTriggeredMode = DISABLE;
 hcan.Init.AutoBusOff        = ENABLE;
 hcan.Init.AutoWakeUp        = ENABLE;
 hcan.Init.AutoRetransmission= ENABLE;
```

```
 hcan.Init.ReceiveFifoLocked = DISABLE;
 hcan.Init.TransmitFifoPriority = DISABLE;
 if (HAL_CAN_Init(&hcan) != HAL_OK) Error_Handler();
}
/* Minimal error handler */
void Error_Handler(void)
{
 __disable_irq();
 while (1) { /* trap here; you can blink LED if desired */ }
}
```

And the Receiver Code

```
#include "main.h"
#include "stm32f0xx_hal.h"
#include <string.h>
#include <stdio.h>
#include <stdarg.h>
/* ---- Handles ---- */
UART_HandleTypeDef huart2;
CAN_HandleTypeDef  hcan;
/* ---- Prototypes ---- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_CAN_Init(void);
/* ---- Tiny printf to UART ---- */
static void uprintf(const char *fmt, ...)
{
 char buf[128];
 va_list ap; va_start(ap, fmt);
 int n = vsnprintf(buf, sizeof(buf), fmt, ap);
 va_end(ap);
 if (n < 0) return;
 if (n > (int)sizeof(buf)) n = sizeof(buf);
 HAL_UART_Transmit(&huart2, (uint8_t*)buf, (uint16_t)n, HAL_MAX_DELAY);
}
/* ======================= MAIN ======================= */
int main(void)
{
 HAL_Init();
 SystemClock_Config();    // 48 MHz
 MX_GPIO_Init();
 MX_USART2_UART_Init();   // VCP → PuTTY
 MX_CAN_Init();           // CAN on PA11/PA12
 /* Configure CAN filter: accept all IDs to FIFO0 */
 CAN_FilterTypeDef filter = {0};
 filter.FilterBank = 0;
 filter.FilterMode = CAN_FILTERMODE_IDMASK;
 filter.FilterScale = CAN_FILTERSCALE_32BIT;
 filter.FilterIdHigh = 0x0000;
 filter.FilterIdLow  = 0x0000;
 filter.FilterMaskIdHigh = 0x0000;
 filter.FilterMaskIdLow  = 0x0000;
 filter.FilterFIFOAssignment = CAN_RX_FIFO0;
 filter.FilterActivation = ENABLE;
 HAL_CAN_ConfigFilter(&hcan, &filter);
 /* Start CAN + enable RX interrupt */
 if (HAL_CAN_Start(&hcan) != HAL_OK) {
   uprintf("CAN start failed\r\n");
   while (1);
 }
 HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING);
 uprintf("CAN ready (500k @ 48MHz). Waiting for frames...\r\n");
 while (1) {
   /* your app can sleep or do other work here */
   HAL_Delay(50);
 }
}
```

```c
/* ============ CAN RX interrupt callback ============ */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan_)
{
 CAN_RxHeaderTypeDef rxh;
 uint8_t data[8];
 if (HAL_CAN_GetRxMessage(hcan_, CAN_RX_FIFO0, &rxh, data) == HAL_OK)
 {
  if (rxh.IDE == CAN_ID_STD) {
    uprintf("ID:0x%03lX ", (uint32_t)rxh.StdId);
  } else {
    uprintf("ID:0x%08lX ", (uint32_t)rxh.ExtId);
  }
  uprintf("DLC:%u DATA:", rxh.DLC);
  for (uint8_t i = 0; i < rxh.DLC; i++) uprintf(" %02X", data[i]);
  uprintf("\r\n");
 }
}
/* ================= Peripheral inits ================= */
/* 48 MHz system clock using HSI48 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /* Enable 48 MHz internal RC */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48;
 RCC_OscInitStruct.HSI48State     = RCC_HSI48_ON;
 RCC_OscInitStruct.PLL.PLLState   = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  Error_Handler();
 /* Set SYSCLK = 48 MHz, HCLK = 48 MHz, PCLK1 = 48 MHz */
 RCC_ClkInitStruct.ClockType      = RCC_CLOCKTYPE_HCLK |
                                    RCC_CLOCKTYPE_SYSCLK |
                                    RCC_CLOCKTYPE_PCLK1;
 RCC_ClkInitStruct.SYSCLKSource   = RCC_SYSCLKSOURCE_HSI48;
 RCC_ClkInitStruct.AHBCLKDivider  = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  Error_Handler();
}
/* GPIO (CubeMX would usually generate this) */
static void MX_GPIO_Init(void)
{
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 /* No manual GPIO needed here; CAN/USART pins get configured by their inits */
}
/* USART2 (Virtual COM, PA2=TX, PA15=RX on Nucleo-32) */
static void MX_USART2_UART_Init(void)
{
 huart2.Instance = USART2;
 huart2.Init.BaudRate   = 9600;
 huart2.Init.WordLength = UART_WORDLENGTH_8B;
 huart2.Init.StopBits   = UART_STOPBITS_1;
 huart2.Init.Parity     = UART_PARITY_NONE;
 huart2.Init.Mode       = UART_MODE_TX_RX;
 huart2.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
 huart2.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart2) != HAL_OK) while (1);
}
/* CAN on PA11(CAN_RX) / PA12(CAN_TX) */
static void MX_CAN_Init(void)
{
 hcan.Instance = CAN;
 hcan.Init.Prescaler = 6;                // 48MHz / (6 * (1+13+2)) = 500 kbps
 hcan.Init.Mode = CAN_MODE_NORMAL;
 hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
 hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
 hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
 hcan.Init.TimeTriggeredMode = DISABLE;
 hcan.Init.AutoBusOff        = ENABLE;
```

```
 hcan.Init.AutoWakeUp       = ENABLE;
 hcan.Init.AutoRetransmission= ENABLE;
 hcan.Init.ReceiveFifoLocked = DISABLE;
 hcan.Init.TransmitFifoPriority = DISABLE;
 if (HAL_CAN_Init(&hcan) != HAL_OK) while (1);
}
/* The F0 maps CAN IRQ to CEC_CAN_IRQn */
void CEC_CAN_IRQHandler(void)
{
 HAL_CAN_IRQHandler(&hcan);
}
/* Simple trap for fatal errors */
void Error_Handler(void)
{
 __disable_irq();
 while (1) {
   /* optional: blink an LED here so you can see it locked up */
 }
}
```

CASEY Reciever CODE:

```
#include "main.h"

CAN_HandleTypeDef hcan;

#define LED_PIN   GPIO_PIN_5    // Nucleo LD2
#define LED_PORT  GPIOA

static void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN_Init(void);

static volatile uint32_t last_rx_tick = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();   // 48 MHz
  MX_GPIO_Init();         // LED + PA11/PA12 AF for CAN
  MX_CAN_Init();          // 500 kbps

  while (1)
  {
    // Blink LED briefly when a frame is received; otherwise keep it off
    if (HAL_GetTick() - last_rx_tick < 100)
      HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_SET);
    else
      HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET);
```

```
  }
}

/* ======= Clock ======= */
static void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48;
  RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();

  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI48;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
Error_Handler();
}

/* ======= GPIO (LED + CAN pins + remap) ======= */
static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_SYSCFG_CLK_ENABLE();

  // LED
  GPIO_InitTypeDef g = {0};
  g.Pin = LED_PIN;
  g.Mode = GPIO_MODE_OUTPUT_PP;
  g.Pull = GPIO_NOPULL;
  g.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_PORT, &g);

  // Remap CAN to PA11/PA12
  SYSCFG->CFGR1 |= SYSCFG_CFGR1_PA11_PA12_RMP;

  // PA11 = CAN_RX, PA12 = CAN_TX (AF4)
  g.Mode = GPIO_MODE_AF_PP;
  g.Pull = GPIO_NOPULL;
  g.Speed = GPIO_SPEED_FREQ_HIGH;
```

```
  g.Pin = GPIO_PIN_11; g.Alternate = GPIO_AF4_CAN; HAL_GPIO_Init(GPIOA, &g);
  g.Pin = GPIO_PIN_12; g.Alternate = GPIO_AF4_CAN; HAL_GPIO_Init(GPIOA, &g);
}

/* ======= CAN init + accept-all filter + IRQ ======= */
static void MX_CAN_Init(void)
{
  __HAL_RCC_CAN1_CLK_ENABLE();

  hcan.Instance = CAN;
  hcan.Init.Prescaler = 6;
  hcan.Init.Mode = CAN_MODE_NORMAL;
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
  hcan.Init.AutoBusOff = ENABLE;
  hcan.Init.AutoWakeUp = ENABLE;
  hcan.Init.AutoRetransmission = ENABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;

  if (HAL_CAN_Init(&hcan) != HAL_OK) Error_Handler();

  CAN_FilterTypeDef f = {0};
  f.FilterBank           = 0;
  f.FilterMode           = CAN_FILTERMODE_IDMASK;
  f.FilterScale          = CAN_FILTERSCALE_32BIT;
  f.FilterIdHigh         = 0x0000;
  f.FilterIdLow          = 0x0000;
  f.FilterMaskIdHigh     = 0x0000;
  f.FilterMaskIdLow      = 0x0000;
  f.FilterFIFOAssignment = CAN_RX_FIFO0;
  f.FilterActivation     = ENABLE;
  HAL_CAN_ConfigFilter(&hcan, &f);

  if (HAL_CAN_Start(&hcan) != HAL_OK) Error_Handler();

  HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING);

  // Enable CAN RX0 IRQ in NVIC
  HAL_NVIC_SetPriority(CEC_CAN_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(CEC_CAN_IRQn);
}
```

```
/* ======= RX callback: timestamp last frame ======= */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *h)
{
  CAN_RxHeaderTypeDef rx;
  uint8_t data[8];
  if (HAL_CAN_GetRxMessage(h, CAN_RX_FIFO0, &rx, data) == HAL_OK)
  {
    last_rx_tick = HAL_GetTick();
  }
}

/* ======= IRQ handler (name for F0's CAN/CEC line) ======= */
void CEC_CAN_IRQHandler(void)
{
  HAL_CAN_IRQHandler(&hcan);
}

/* ======= Minimal error handler ======= */
void Error_Handler(void)
{
  __disable_irq();
  while (1) { }
}
```