DESN2000 Debugging Guide

- Common Problems
  - **No STLink device connected**
  - **Failed to start GDB server Error in initialising ST-Link device**
  - **Missing ST-Link drivers**
- Debugging Execution
  - Enabling Debugging
  - Adding Code
  - Intro to Debug mode, resume, suspend, and watch variables
  - Breakpoints
  - The Debugger SWO
  - **The ST-Link Virtual COM Port**

Common Problems

No STLink device connected

- Make sure you're logged into the IDE (should say "Hello NAME" in the top bar instead of 'myST")
- An upgrade window may appear saying that the st link is out of date
  - Click 'open in upgrade mode', wait a few seconds
  - Click 'upgrade' button down the bottom and wait for it to finish - once finished you can try and run your program again

Failed to start GDB server Error in initialising ST-Link device

- Make sure you're logged into the IDE (should say "Hello NAME" in the top bar instead of 'myST")
- An upgrade window may appear saying that the st link is out of date
  - Click 'open in upgrade mode', wait a few seconds
  - Click 'upgrade' button down the bottom and wait for it to finish - once finished you can try and run your program again

Missing ST-Link drivers

- Go to this website here, or just download the zip file below and install the drivers
  - Click the .msi file if you use windows
  - Click the .pkg file if you use mac

    en.st-link-server-v2-1-1.zip

- You may then get asked to upgrade your st link after you install the drivers - follow the steps in the **Failed to start GDB server Error in initialising ST-Link device**

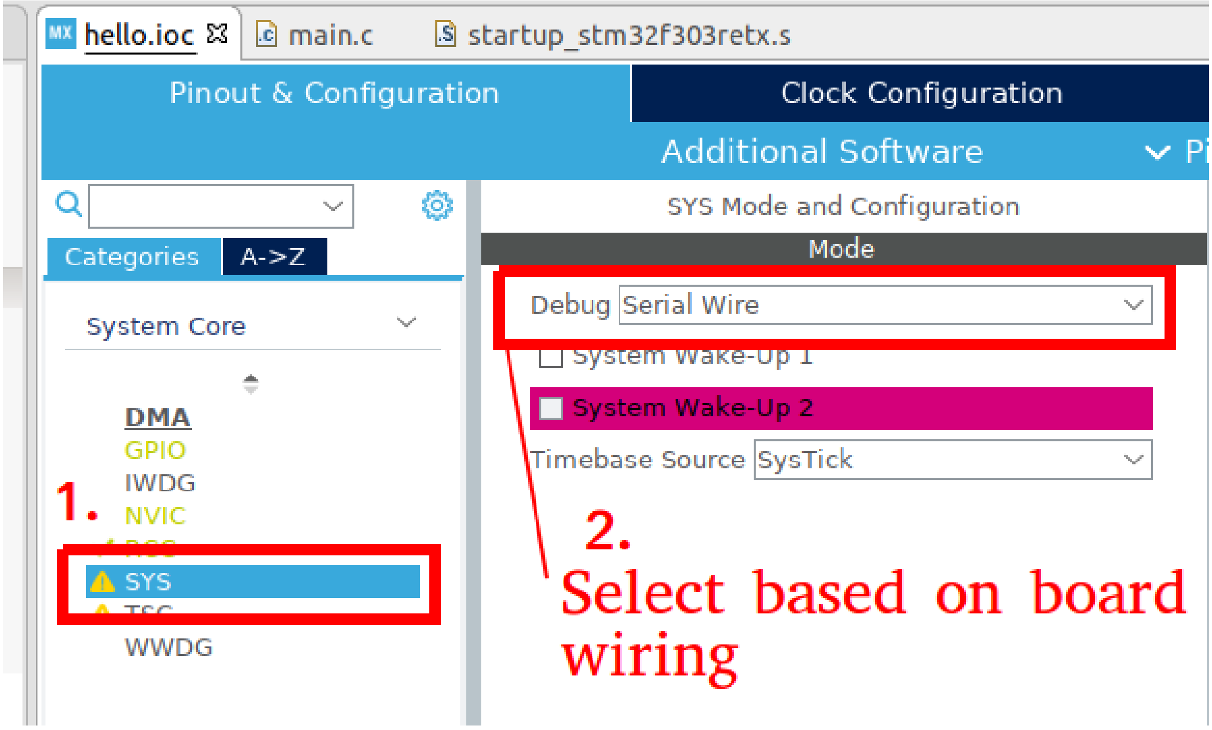
Debugging Execution

In this tutorial we will focuses on software debugging, with breakpoints, watch lists, trace analysers, and so forth. STM32CubeIDE has a powerful integrated debugger.

This tutorial is based off of Hammond’s found here.

Enabling Debugging

Note: You may need to enable debugging via your CubeMX config. I had it enabled already since it’s included in the default configuration for my development board. If you need to enable it, do this in the CubeMX view:



Debug Serial Wire

Timebase Source SysTick

1.

2. Select based on board wiring

Save your config using `File>Save` and regenerate the code.

Adding Code

Have a play by debugging the code. But first, add something to debug, since there's not much going on here. Add a few snippets of code to calculate some prime numbers.

Add a few lines of code to a few of those USER code blocks in main.c, as detailed here:

```
// . . .

/* USER CODE BEGIN Includes */
#include <stdbool.h>
/* USER CODE END Includes */

// . . .

/* USER CODE BEGIN PD */
```

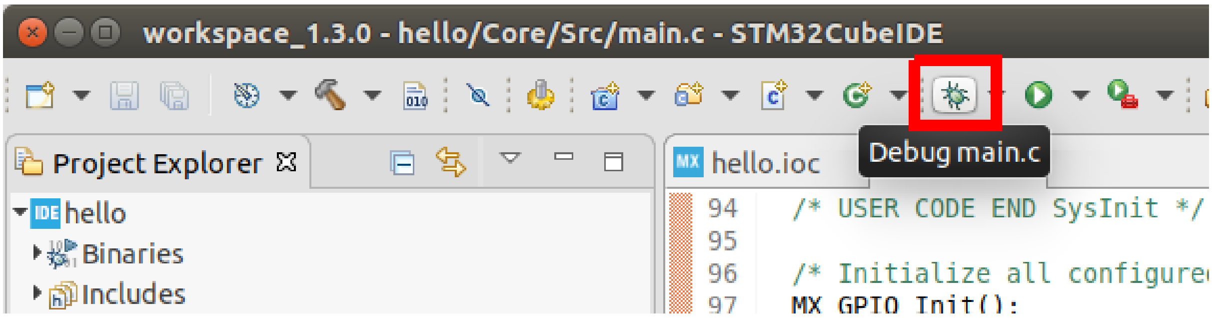
```
#define PRIMES_LEN 62
/* USER CODE END PD */

// . . .

/* USER CODE BEGIN PV */
uint16_t primes[PRIMES_LEN] = {0};
/* USER CODE END PV */

// . . .

/* USER CODE BEGIN 0 */

//check if a number is prime
bool is_prime(uint16_t v) {
  for(uint16_t i=2; i<(v/2 + 1); i++) {
    if(v % i == 0) return false;
  }
  return true;
}
/* USER CODE END 0 */

// . . . (inside int main())

  /* USER CODE BEGIN 2 */

  //calculate a list of primes:
  uint16_t prime_index = 0;

  for(uint16_t i = 2; i < 300; i++) {
    if(is_prime(i)) {
      primes[prime_index] = i;
      prime_index++;
    }
  }

  /* USER CODE END 2 */
```

Compile and execute this; there won't be any observable changes (except for maybe the slightest delay before the LED starts blinking the first time).

First, look at the end result, then observe the looping calculation using a breakpoint.

Intro to Debug mode, resume, suspend, and watch variables

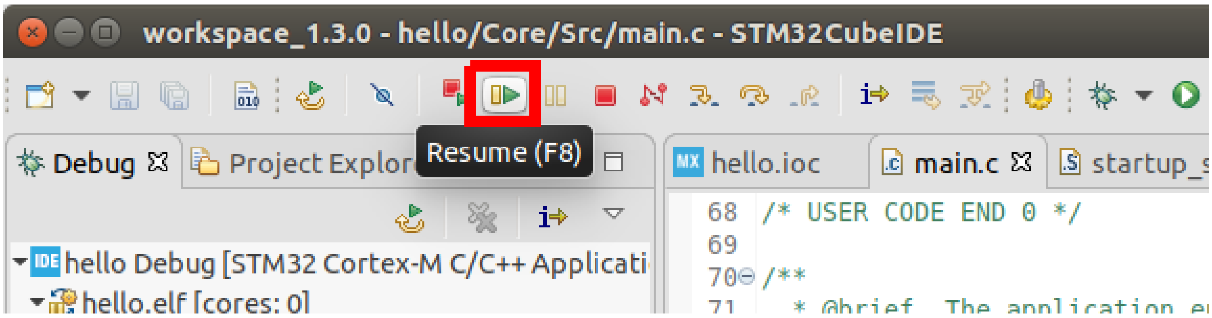
To simply observe the end result, we launch the program using the Debug mode, with the **Debug** button.



```
/* USER CODE END SysInit */

/* Initialize all configure
MX_GPIO_Init();
```

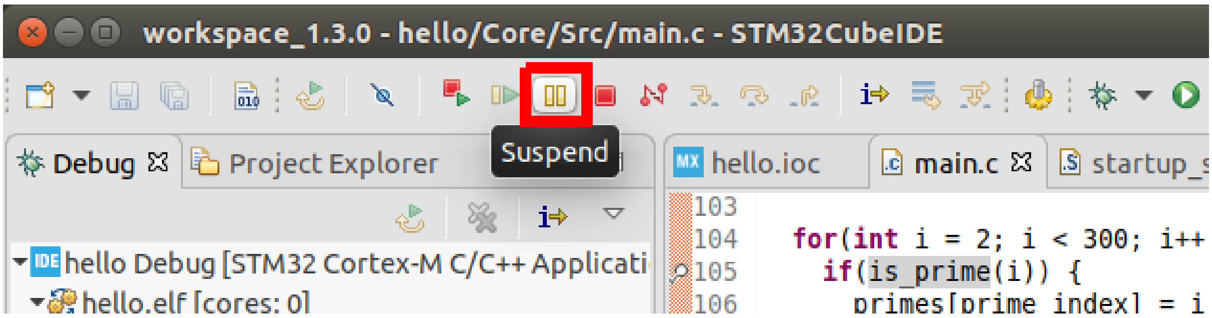
Firstly, STM32CubeIDE will change into its *debug* perspective. There may be a popup asking about this.

Notice that the dev kit in front of you has the communication LED blinking continuously, but the user LED isn't blinking yet. This is because the program is not actually running yet. To run the program, press the 'Resume' key to get started. The resume key, as well as the other debug control keys, will have appeared due to the *debug* perspective activating.



```
/* USER CODE END 0 */

/**
  * @brief  The application e
```

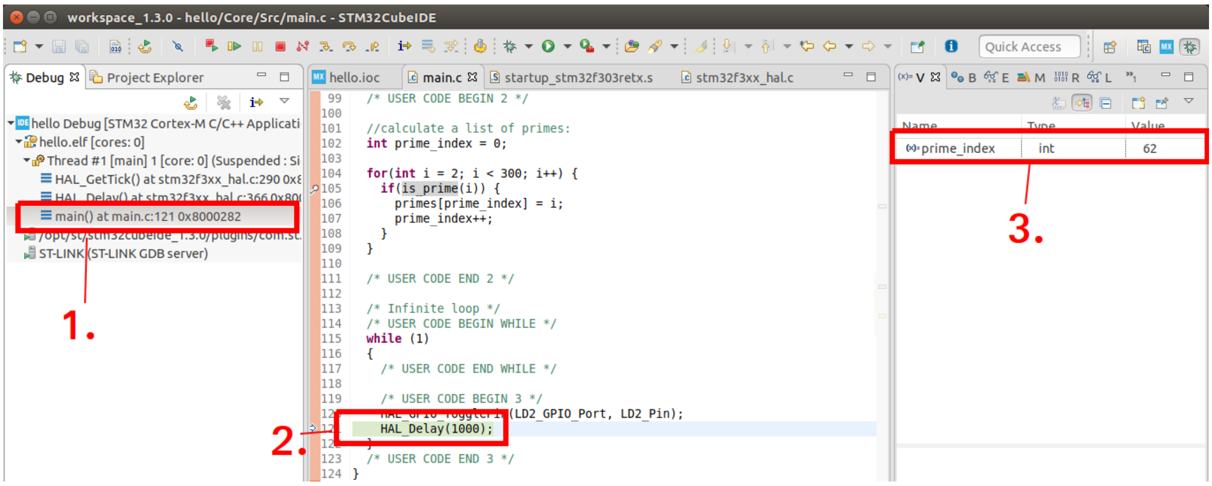
Once you notice the dev board’s LED light is blinking, pause the execution by pressing the suspend key:



```
  for(int i = 2; i < 300; i++
    if(is_prime(i)) {
      primes[prime_index] = i
```

Notice that the IDE immediately throws you somewhere in the C code for your project, almost certainly in something to do with the HAL_Delay function. That's where the program was when suspend was hit.

Get back home by going to the left Debug panel and selecting `main()`.

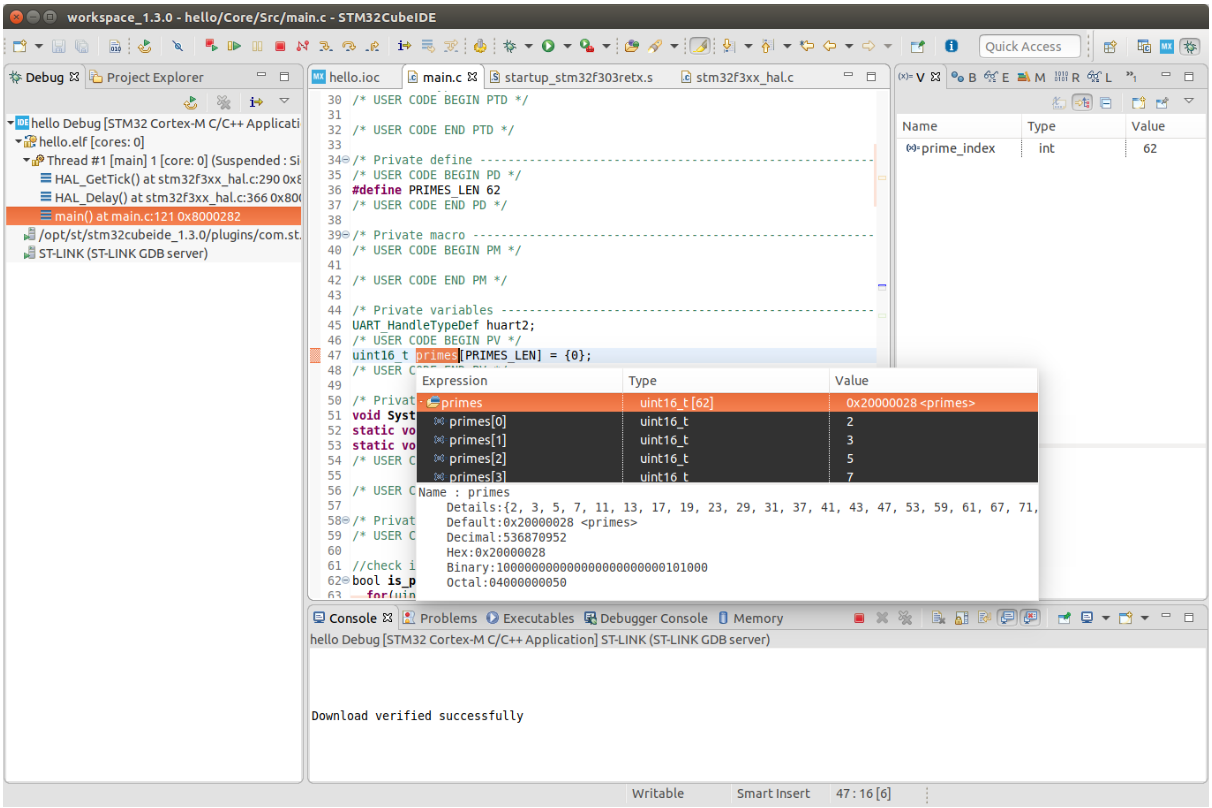


```
  /* USER CODE BEGIN 2 */

  //calculate a list of primes:
  int prime_index = 0;

  for(int i = 2; i < 300; i++) {
    if(is_prime(i)) {
      primes[prime_index] = i;
      prime_index++;
    }
  }

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
```

Name	Type	Value
prime_index	int	62

Note that the IDE highlights the function that is currently being executed (2) and presents a list of variables in the current scope (3).

Examine the contents of the global variable by double-clicking it.



```
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define PRIMES_LEN 62
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;
/* USER CODE BEGIN PV */
uint16_t primes[PRIMES_LEN] = {0};
```

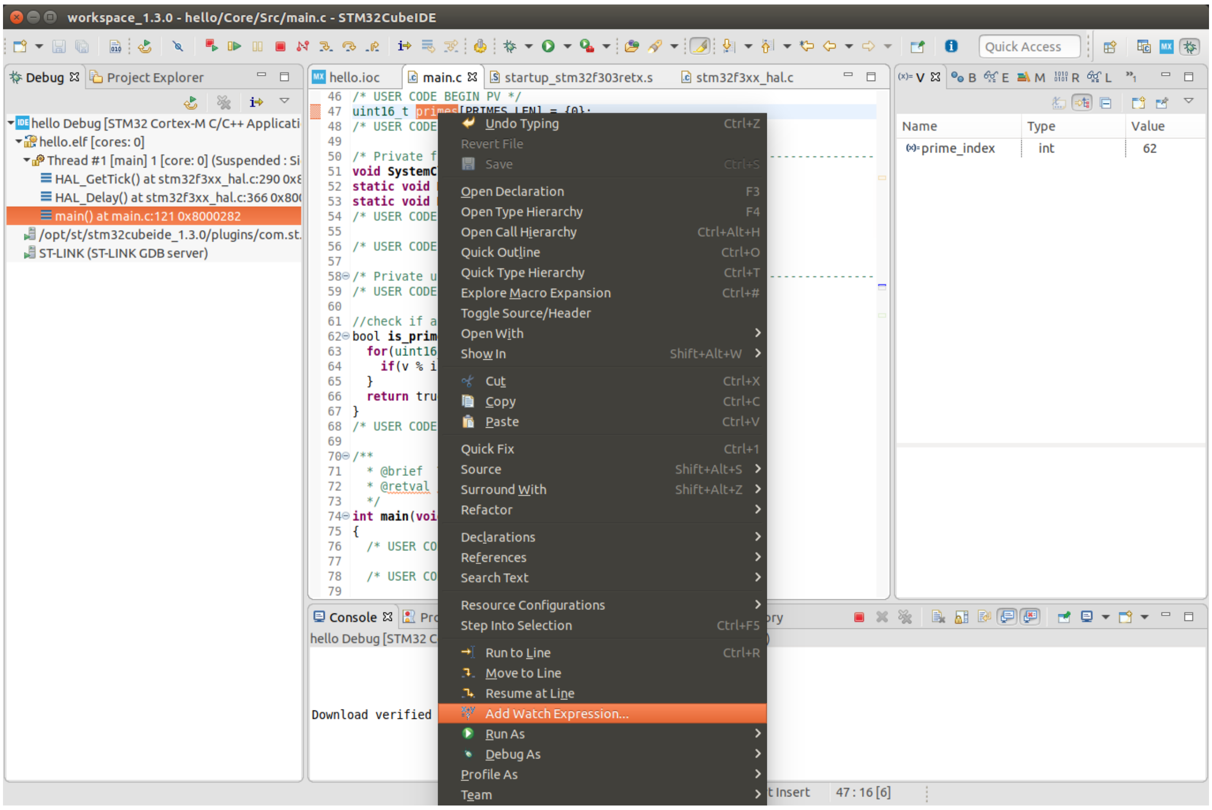
Name	Type	Value
prime_index	int	62

Expression	Type	Value
primes	uint16_t [62]	0x20000028 <primes>
primes[0]	uint16_t	2
primes[1]	uint16_t	3
primes[2]	uint16_t	5
primes[3]	uint16_t	7

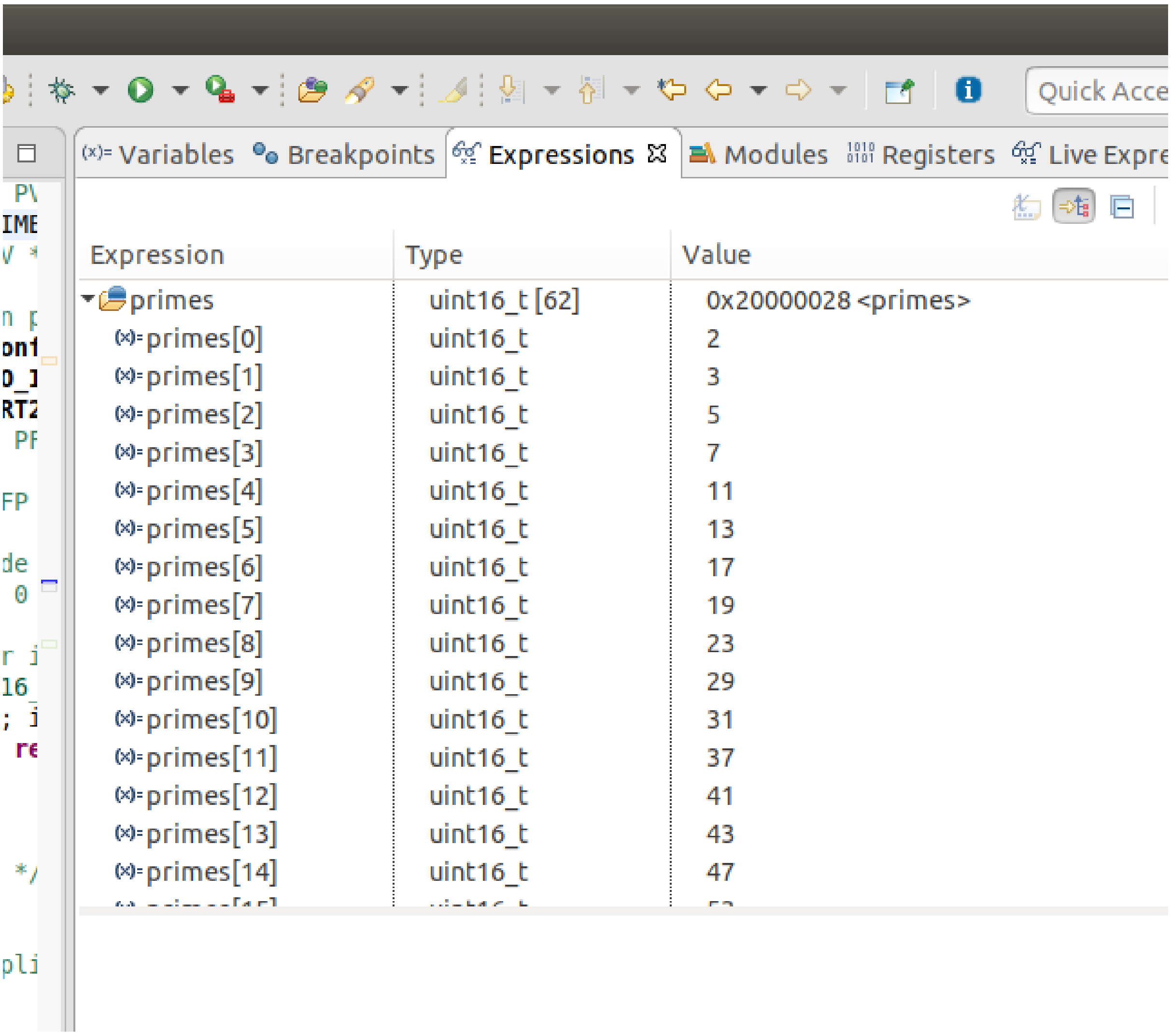
```
Name : primes
    Details:{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
    Default:0x20000028 <primes>
    Decimal:536870952
    Hex:0x20000028
    Binary:100000000000000000000000101000
    Octal:04000000050
```

```
hello Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server)

Download verified successfully
```

Or in a more convenient manner (since this is a big array) by right clicking on the variable `primes` and selecting `Add Watch Expression` (and then pressing OK)



That then adds it to the `Expressions` menu on the right:



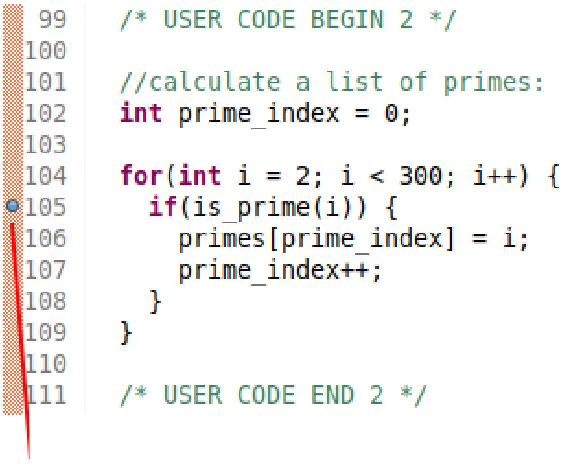
Expression	Type	Value
primes	uint16_t [62]	0x20000028 <primes>
primes[0]	uint16_t	2
primes[1]	uint16_t	3
primes[2]	uint16_t	5
primes[3]	uint16_t	7
primes[4]	uint16_t	11
primes[5]	uint16_t	13
primes[6]	uint16_t	17
primes[7]	uint16_t	19
primes[8]	uint16_t	23
primes[9]	uint16_t	29
primes[10]	uint16_t	31
primes[11]	uint16_t	37
primes[12]	uint16_t	41
primes[13]	uint16_t	43
primes[14]	uint16_t	47

Press the Red Stop button now in the top menu, and watch the execution of the prime calculation loop.

Breakpoints

In addition to running and suspending execution, we can also ask the program to suspend at a point of our choosing. This is known as creating a breakpoint.

In STM32CubeIDE, you do this by double-clicking on the red bar next to the line numbers, which will cause a small blue breakpoint indicator dot to appear.



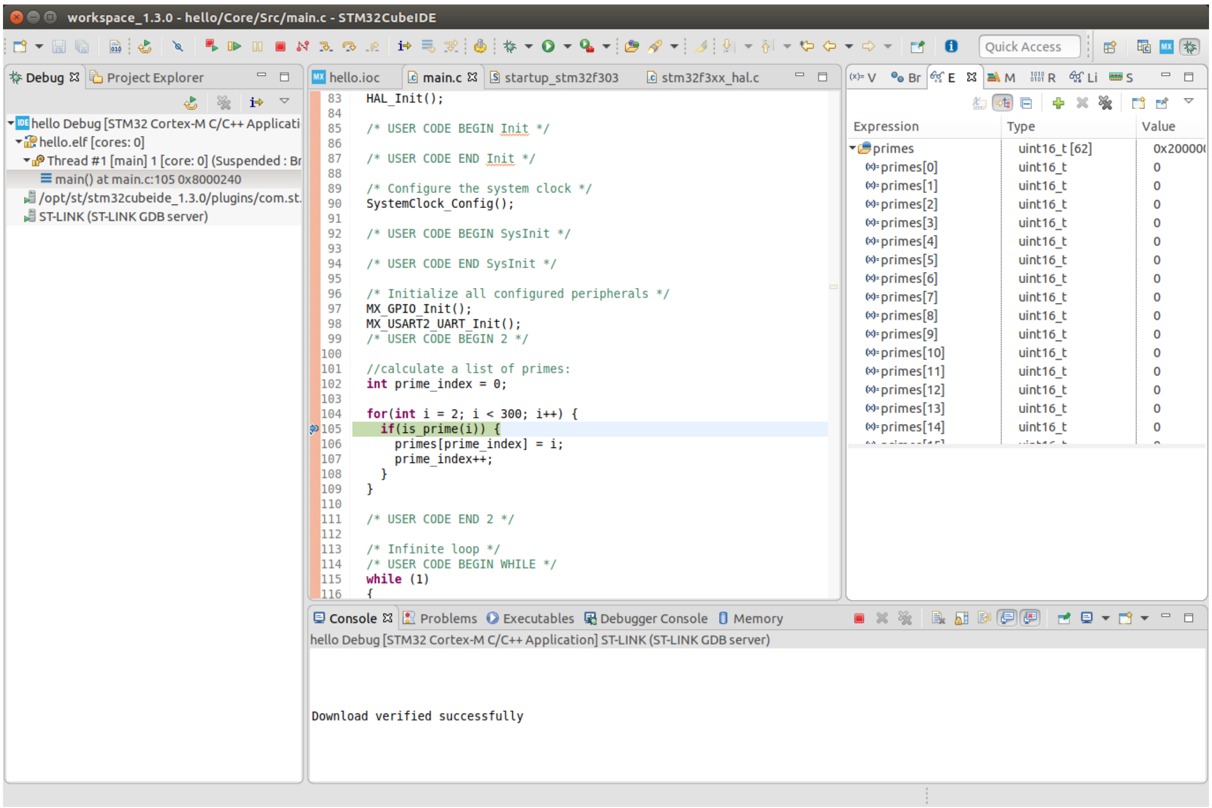
```
/* USER CODE BEGIN 2 */

//calculate a list of primes:
int prime_index = 0;

for(int i = 2; i < 300; i++) {
  if(is_prime(i)) {
    primes[prime_index] = i;
    prime_index++;
  }
}

/* USER CODE END 2 */
```

Double click on the red bar next to the line numbers.

A blue breakpoint dot will appear.

Now, without changing anything else, launch the debug mode again.

This time, when you hit resume, you'll notice that the program executes and then automatically halts when it reaches your breakpoint.



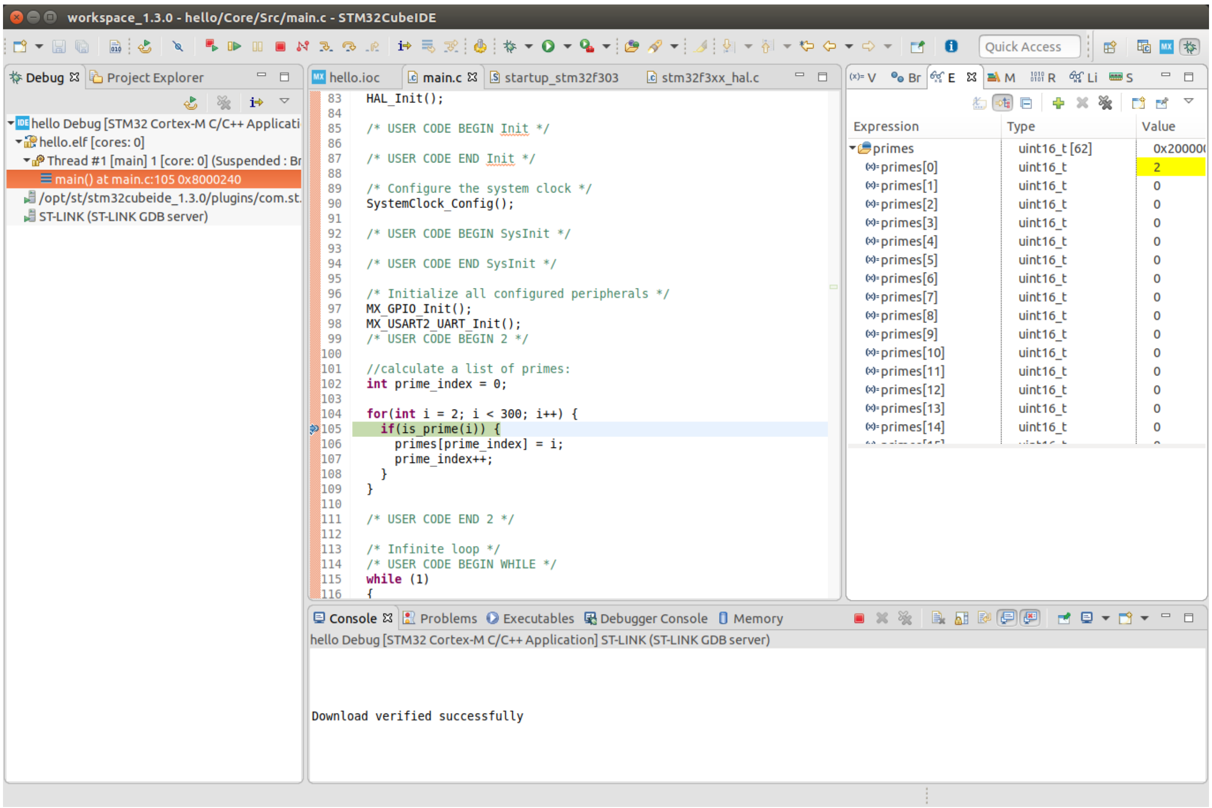
hello Debug [STM32 Cortex-M C/C++ Applicati
hello.elf [cores: 0]
Thread #1 [main] 1 [core: 0] (Suspended : Br
main() at main.c:105 0x8000240
/opt/st/stm32cubeide_1.3.0/plugins/com.st.
ST-LINK (ST-LINK GDB server)

```
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */

//calculate a list of primes:
int prime_index = 0;

for(int i = 2; i < 300; i++) {
  if(is_prime(i)) {
    primes[prime_index] = i;
    prime_index++;
  }
}

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
```

Expression	Type	Value
primes	uint16_t [62]	0x20000(
primes[0]	uint16_t	0
primes[1]	uint16_t	0
primes[2]	uint16_t	0
primes[3]	uint16_t	0
primes[4]	uint16_t	0
primes[5]	uint16_t	0
primes[6]	uint16_t	0
primes[7]	uint16_t	0
primes[8]	uint16_t	0
primes[9]	uint16_t	0
primes[10]	uint16_t	0
primes[11]	uint16_t	0
primes[12]	uint16_t	0
primes[13]	uint16_t	0
primes[14]	uint16_t	0

```
hello Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server)


Download verified successfully
```

Now if press suspend again, it will loop and stop at this same function.



hello Debug [STM32 Cortex-M C/C++ Applicati
hello.elf [cores: 0]
Thread #1 [main] 1 [core: 0] (Suspended : Br
main() at main.c:105 0x8000240
/opt/st/stm32cubeide_1.3.0/plugins/com.st.
ST-LINK (ST-LINK GDB server)

```
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  //calculate a list of primes:
  int prime_index = 0;

  for(int i = 2; i < 300; i++) {
    if(is_prime(i)) {
      primes[prime_index] = i;
      prime_index++;
    }
  }

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

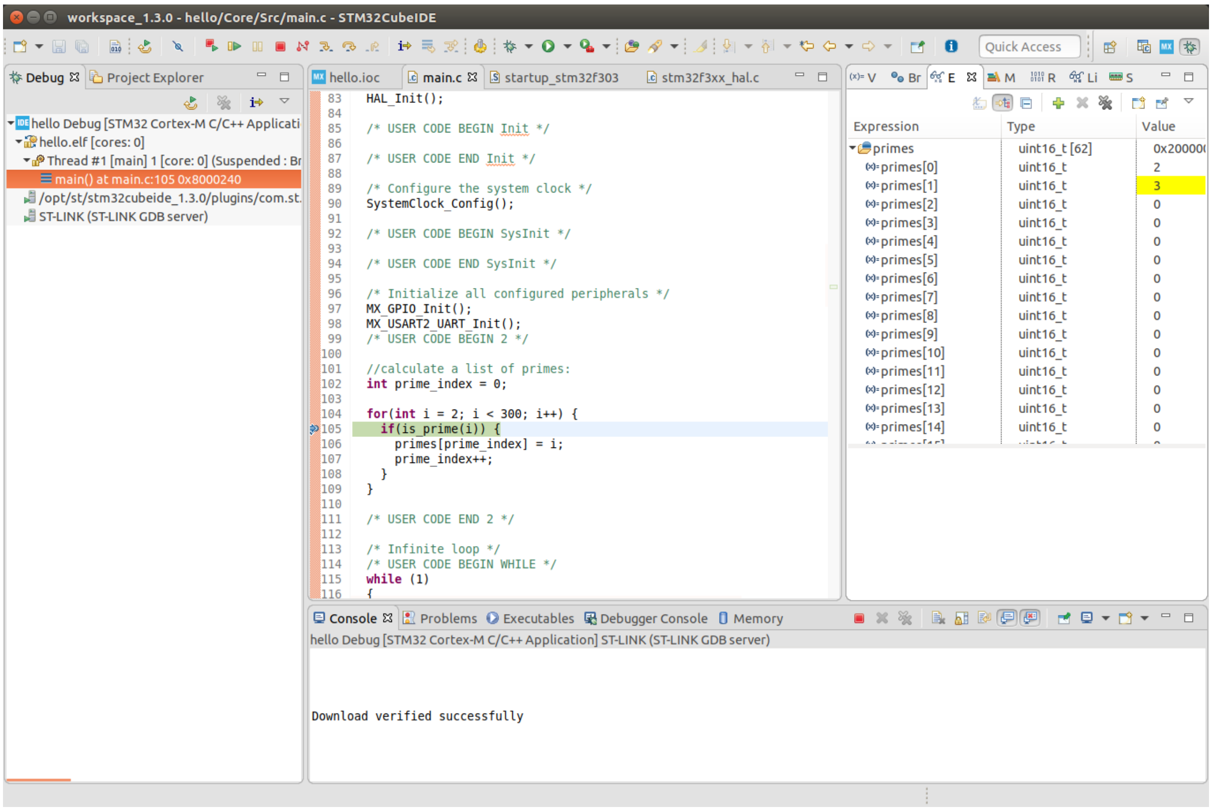
Expression	Type	Value
primes	uint16_t [62]	0x20000(
primes[0]	uint16_t	2
primes[1]	uint16_t	0
primes[2]	uint16_t	0
primes[3]	uint16_t	0
primes[4]	uint16_t	0
primes[5]	uint16_t	0
primes[6]	uint16_t	0
primes[7]	uint16_t	0
primes[8]	uint16_t	0
primes[9]	uint16_t	0
primes[10]	uint16_t	0
primes[11]	uint16_t	0
primes[12]	uint16_t	0
primes[13]	uint16_t	0
primes[14]	uint16_t	0

hello Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server)

```
Download verified successfully
```

The first index of primes has changed, and it’s highlighted the changed variable.

Hitting resume again:



```
hello Debug [STM32 Cortex-M C/C++ Applicati
  hello.elf [cores: 0]
    Thread #1 [main] 1 [core: 0] (Suspended : Br
      main() at main.c:105 0x8000240
  /opt/st/stm32cubeide_1.3.0/plugins/com.st.
  ST-LINK (ST-LINK GDB server)
```

```c
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  //calculate a list of primes:
  int prime_index = 0;

  for(int i = 2; i < 300; i++) {
    if(is_prime(i)) {
      primes[prime_index] = i;
      prime_index++;
    }
  }

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

Expression	Type	Value
primes	uint16_t [62]	0x20000(
primes[0]	uint16_t	2
primes[1]	uint16_t	3
primes[2]	uint16_t	0
primes[3]	uint16_t	0
primes[4]	uint16_t	0
primes[5]	uint16_t	0
primes[6]	uint16_t	0
primes[7]	uint16_t	0
primes[8]	uint16_t	0
primes[9]	uint16_t	0
primes[10]	uint16_t	0
primes[11]	uint16_t	0
primes[12]	uint16_t	0
primes[13]	uint16_t	0
primes[14]	uint16_t	0

```
hello Debug [STM32 Cortex-M C/C++ Application] ST-LINK (ST-LINK GDB server)

Download verified successfully
```

Now the second index of primes has changed, and the changed variable is highlighted once more!

Keep pressing the resume button, and you'll see it slowly calculate the array.

Now, let's quit the debugger and move on. You can delete the breakpoint again by double-clicking the blue dot. Note that if you right-click on the red column, you can also toggle and create breakpoints this way. This also brings up advanced breakpoint options, including breakpoint conditions and breakpoint types.

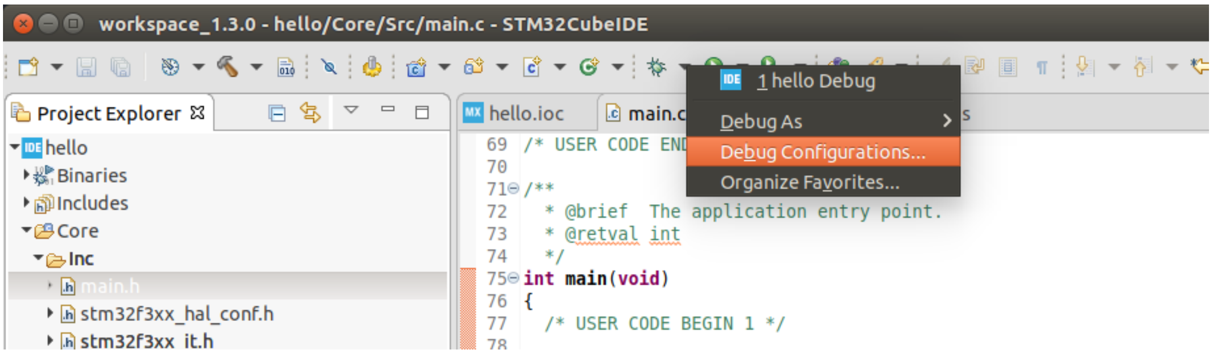
The Debugger SWO

If properly configured, you can output arbitrary strings directly to the debugger via the programmer, rather than sending them via any other peripherals. It's a bit like a virtual UART that you can send data to.

This is a little involved to set up, and it can be worth simply using a UART if you must send out strings of characters to help your debugging, but I'll step through the basics here using STM32CubeIDE.

First, we must configure the reception clock rate. This is done via the debug configuration menu.

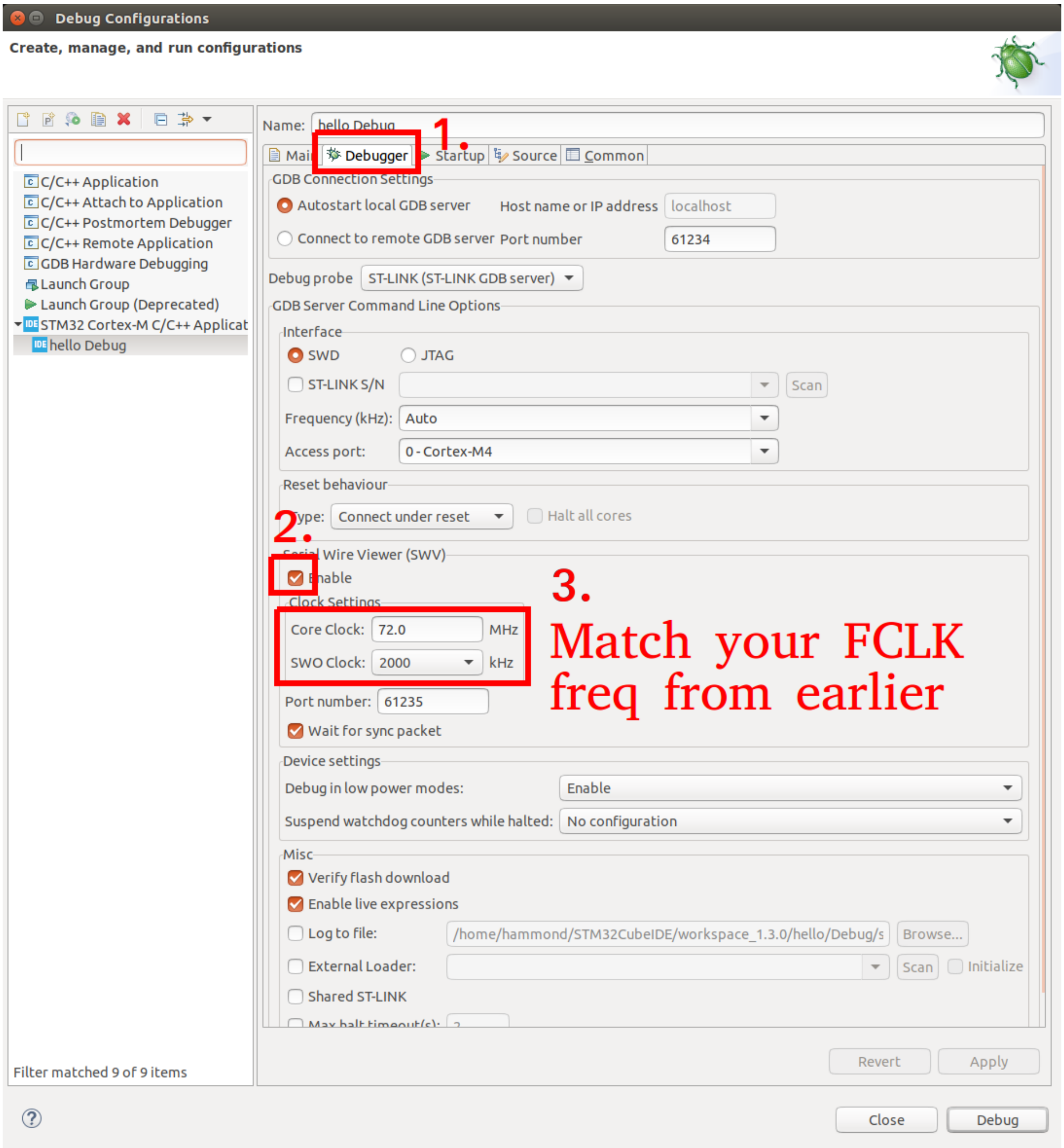
Press this:



```
/* USER CODE END

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
```

Go to the Debugger tab, Enable SWV (Serial Wire Viewer), then change your clock rate to the FCLK from earlier (remember when we chose the clock rate for all of our components?)

You may leave the SWO Clock dropdown set to its maximum.



Name: hello Debug

1.

GDB Connection Settings

● Autostart local GDB server Host name or IP address: localhost

○ Connect to remote GDB server Port number: 61234

Debug probe: ST-LINK (ST-LINK GDB server)

GDB Server Command Line Options

Interface

● SWD ○ JTAG

☐ ST-LINK S/N

Frequency (kHz): Auto

Access port: 0 - Cortex-M4

Reset behaviour

Type: Connect under reset ☐ Halt all cores

2.

Serial Wire Viewer (SWV)

☑ Enable

Clock Settings

Core Clock: 72.0 MHz

SWO Clock: 2000 kHz

Port number: 61235

☑ Wait for sync packet

3.

Match your FCLK freq from earlier

Device settings

Debug in low power modes: Enable

Suspend watchdog counters while halted: No configuration

Misc

☑ Verify flash download

☑ Enable live expressions

☐ Log to file: /home/hammond/STM32CubeIDE/workspace_1.3.0/hello/Debug/s

☐ External Loader: ☐ Initialize

☐ Shared ST-LINK

☐ Max halt timeout(s): 2

Filter matched 9 of 9 items

Now press Apply/Close.

Add a test to send some characters. In the main loop, add the following:

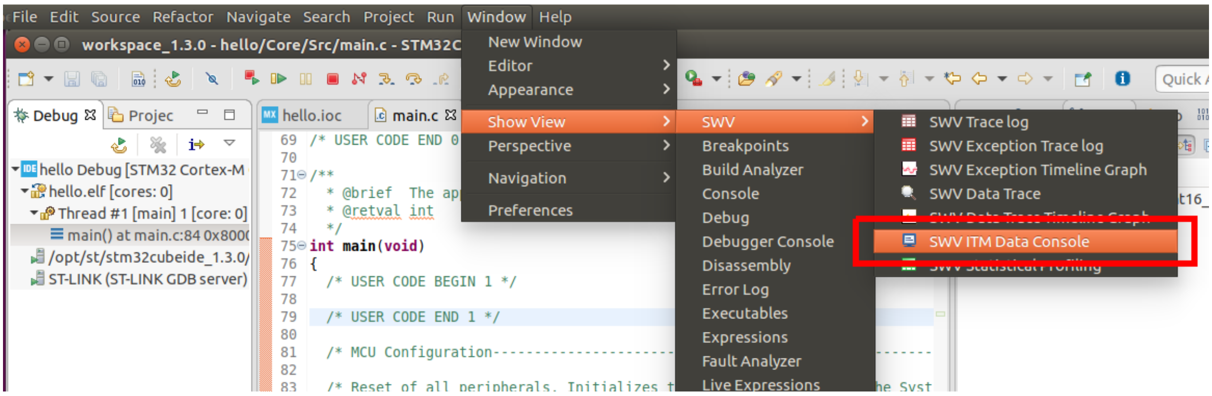
```
 /* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
```

```
    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    HAL_Delay(1000);
    ITM_SendChar('!');
  }
  /* USER CODE END 3 */
```

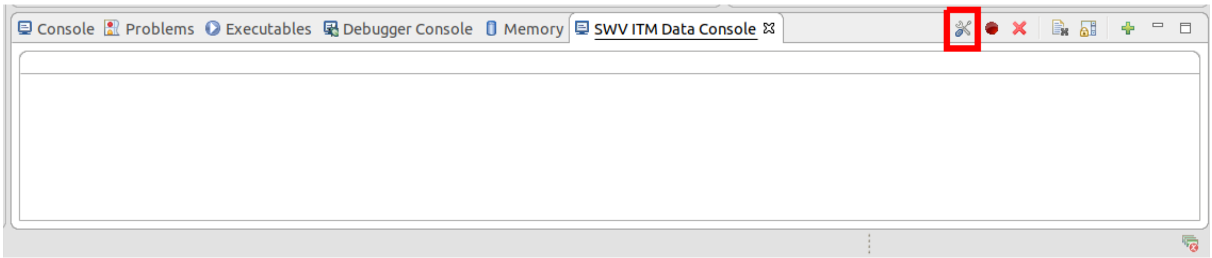
`ITM_SendChar` is a special function that sends a character to the debugger's serial viewer. You shouldn't need to `#include` anything new or special to use this.

You can now launch the debug session as before. But, before you press `Resume` to get it started, we need to enable a few more debugging options.

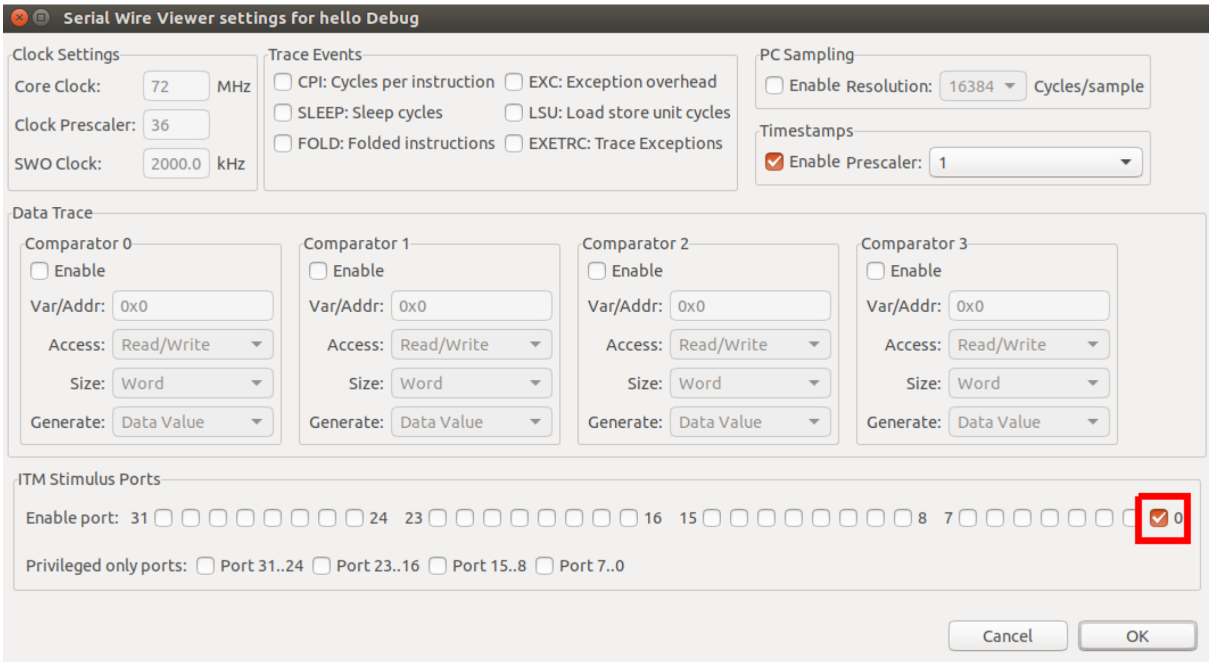
First, open the ITM data console through the `Window` menu.



This will bring up the ITM data console window. Now, enter the configuration menu:



Enable ITM stimulus port 0:



Serial Wire Viewer settings for hello Debug

Clock Settings
Core Clock: 72 MHz
Clock Prescaler: 36
SWO Clock: 2000.0 kHz

Trace Events
☐ CPI: Cycles per instruction
☐ EXC: Exception overhead
☐ SLEEP: Sleep cycles
☐ LSU: Load store unit cycles
☐ FOLD: Folded instructions
☐ EXETRC: Trace Exceptions

PC Sampling
☐ Enable Resolution: 16384 Cycles/sample

Timestamps
☑ Enable Prescaler: 1

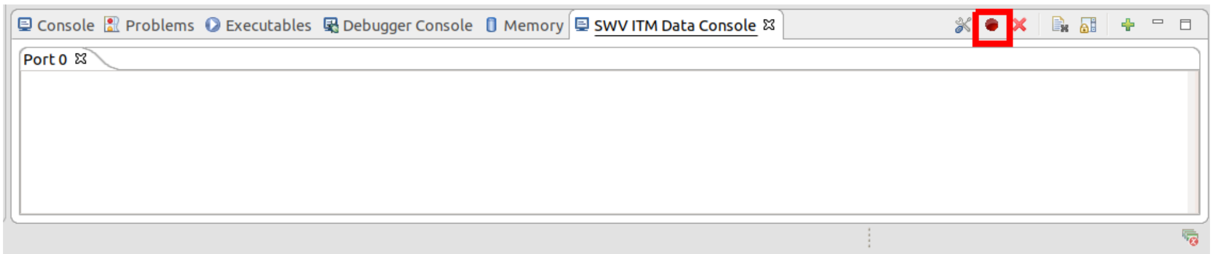
Data Trace

Comparator 0	Comparator 1	Comparator 2	Comparator 3
☐ Enable	☐ Enable	☐ Enable	☐ Enable
Var/Addr: 0x0	Var/Addr: 0x0	Var/Addr: 0x0	Var/Addr: 0x0
Access: Read/Write	Access: Read/Write	Access: Read/Write	Access: Read/Write
Size: Word	Size: Word	Size: Word	Size: Word
Generate: Data Value	Generate: Data Value	Generate: Data Value	Generate: Data Value

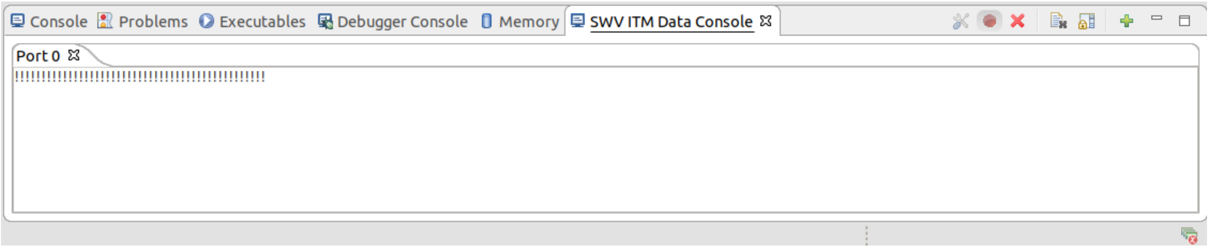
ITM Stimulus Ports
Enable port: 31 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ 24 23 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ 16 15 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ 8 7 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☑ 0
Privileged only ports: ☐ Port 31..24 ☐ Port 23..16 ☐ Port 15..8 ☐ Port 7..0

Press OK.

Port 0 will appear in the console view. Now press `Start` - this contacts your microcontroller and adjusts some registers internally to enable this mechanism:



Now, and only now, can you press Resume. You'll notice your Port 0 terminal slowly start filling with exclamation marks (since that's the character we're sending).



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

As you can imagine, this is pretty handy when your design might not have a free UART for debugging.

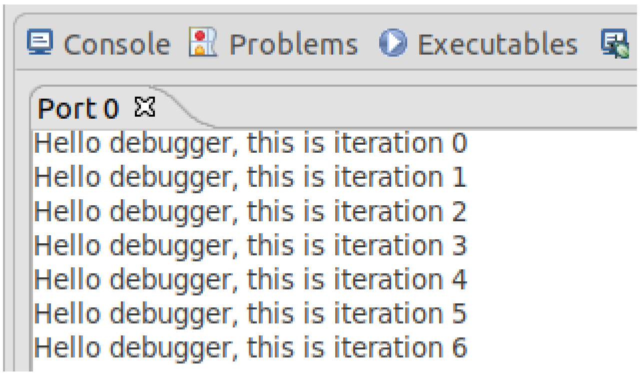
Without too much difficulty, we can also spin up a custom `printf` function for debugging. There's a few options for this, but the preferred approach is to actually create a `debug_printf` function, like so:

Adding more to `main.c`, as detailed:

```
// . . .

/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include <stdio.h>
#include <stdarg.h>
/* USER CODE END Includes */

// . . .

/* USER CODE BEGIN PD */
#define PRIMES_LEN 62
/* USER CODE END PD */

// . . .

/* USER CODE BEGIN PV */
uint16_t primes[PRIMES_LEN] = {0};
/* USER CODE END PV */

// . . .

/* USER CODE BEGIN 0 */

//check if a number is prime
bool is_prime(uint16_t v) {
  // . . .
}

// debug_printf sends a max of 256 characters to the ITM SWO
// It uses a _variable length argument_, same as normal print
// Call this function as if it was printf
void debug_printf(const char *fmt, ...) {
```

```
  char buffer[256];
  va_list args;
  va_start(args, fmt);
  vsnprintf(buffer, sizeof(buffer), fmt, args);
  va_end(args);

  uint16_t i = 0;
  while(buffer[i] != '\0') {
    ITM_SendChar(buffer[i]);
    i++;
  }

}
/* USER CODE END 0 */

// . . . (inside int main())

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  uint32_t count = 0;
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    HAL_Delay(1000);
    debug_printf("Hello debugger, this is iteration %d\r\n",
    count++;
  }
  /* USER CODE END 3 */
```

Now we save, build, and `debug` that. Remember to `Start` the ITM trace before pressing `Resume`!



```
Hello debugger, this is iteration 0
Hello debugger, this is iteration 1
Hello debugger, this is iteration 2
Hello debugger, this is iteration 3
Hello debugger, this is iteration 4
Hello debugger, this is iteration 5
Hello debugger, this is iteration 6
```

The ST-Link Virtual COM Port

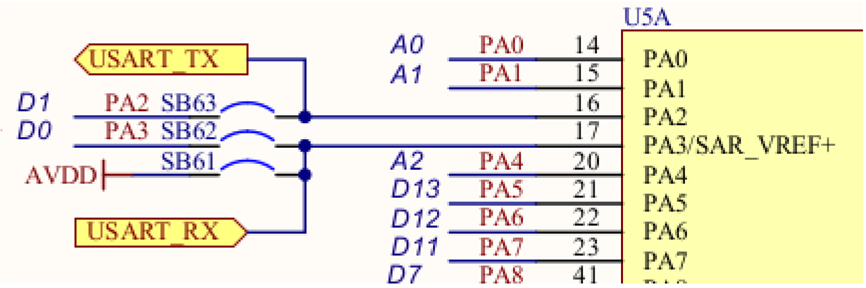
The inbuilt ST-Link v2.1 interface that we've been using for programming and debugging *also* includes a *virtual COM port*. The COM port uses drivers which are included natively in most operating systems (including Windows and Ubuntu Linux).

Running `dmesg | grep tty` in the terminal, we can see it has been made available as `/dev/ttyACM0` thus:

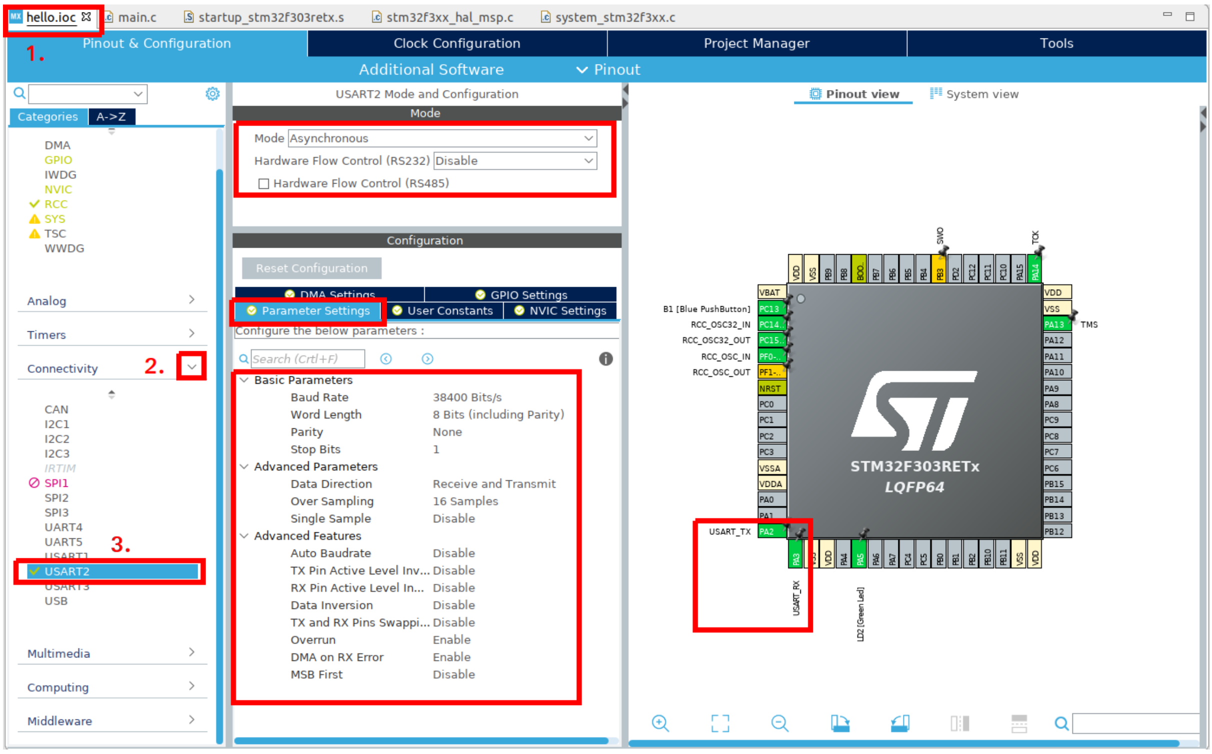
```
$ dmesg | grep tty
[30318.354183] cdc_acm 3-10.4:1.2: ttyACM0: USB ACM device
```

This is really handy for your user applications, as this virtual COM port is wired directly onto one of the USART peripherals on the nucleo board! Recall from the CubeMX view that pins *PA2* and *PA3* were automatically configured as a USART for us.

A quick sanity check to make sure this makes sense by looking on the schematic:



Sure looks like they're connected to a USART (tracing it through the rest of the schematic shows them connected to the ST-Link V2 programmer). So let's quickly jump back into the CubeMX view and see how the port was set up:



Mode: Asynchronous
Hardware Flow Control (RS232): Disable
☐ Hardware Flow Control (RS485)

Basic Parameters
- Baud Rate: 38400 Bits/s
- Word Length: 8 Bits (including Parity)
- Parity: None
- Stop Bits: 1

Advanced Parameters
- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples
- Single Sample: Disable

Advanced Features
- Auto Baudrate: Disable
- TX Pin Active Level Inv...: Disable
- RX Pin Active Level In...: Disable
- Data Inversion: Disable
- TX and RX Pins Swappi...: Disable
- Overrun: Enable
- DMA on RX Error: Enable
- MSB First: Disable

It’s configured as asynchronous, at 38400 baud, 8 data bits, no parity, 1 stop bit.

We could change these settings now if we wanted to. The virtual COM port works the same as any other USB to serial adapter, and so any baud rate and config can work with it.

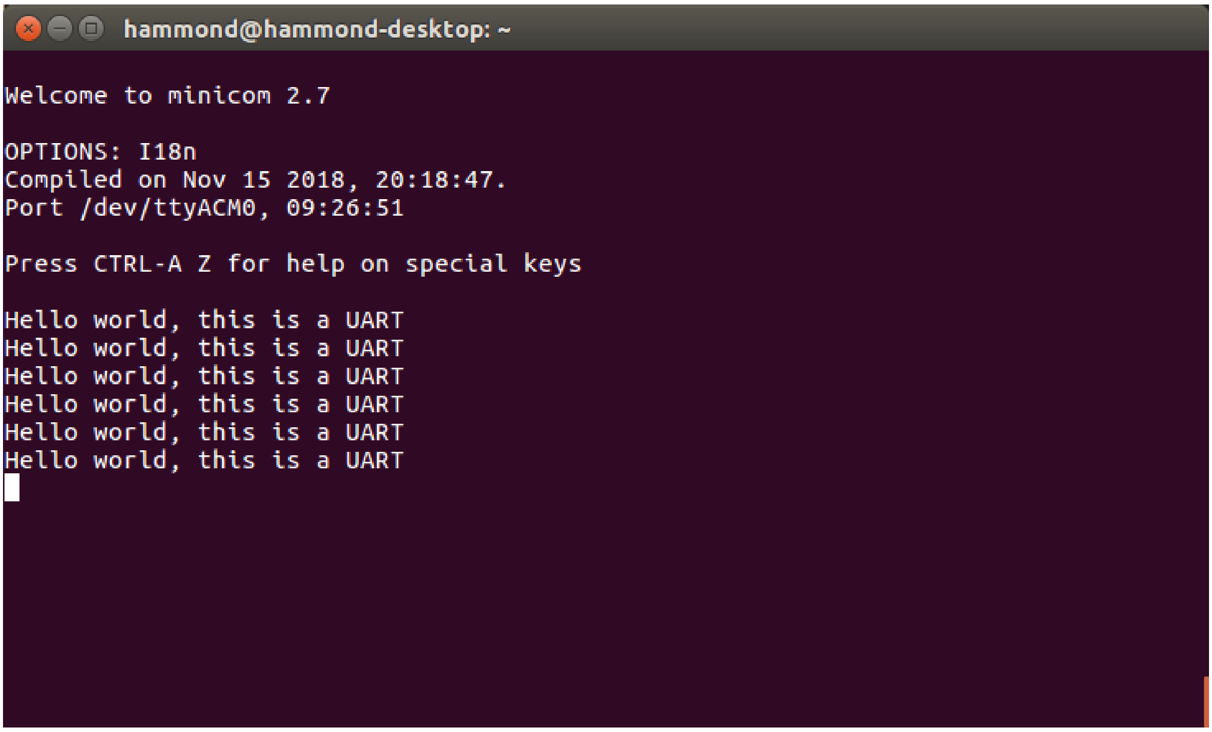
Let’s send some characters to it.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
uint32_t count = 0;
while (1)
{
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
        HAL_Delay(1000);
        debug_printf("Hello debugger, this is iteration %d\r\
        HAL_UART_Transmit(&huart2, (uint8_t*)"Hello world, th
```

```
        count++;
}
/* USER CODE END 3 */
```

We now build and download that, before running Minicom:

```
$ minicom -b 38400 -D /dev/ttyACM0
```



```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Nov 15 2018, 20:18:47.
Port /dev/ttyACM0, 09:26:51

Press CTRL-A Z for help on special keys

Hello world, this is a UART
Hello world, this is a UART
Hello world, this is a UART
Hello world, this is a UART
Hello world, this is a UART
Hello world, this is a UART
```

If we wanted, we could now make a `usart_printf` like our `debug_printf` from earlier.