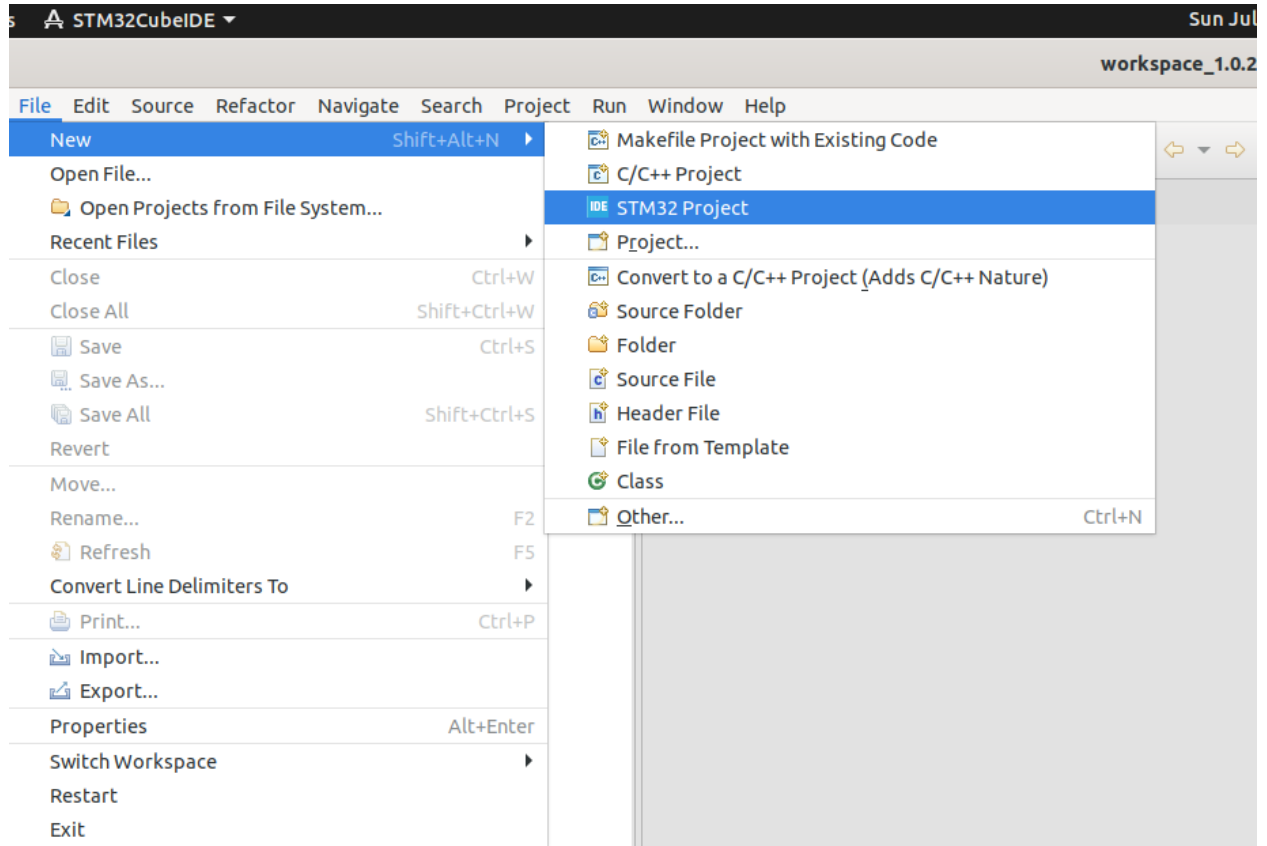


STM32CubeIDE + GL Starter Kit

Getting started

1. Создание нового проекта

В STM32CubeIDE *File* -> *New* -> *STM32 Project*



В *Target Selector* во вкладке *MCU/MPU Selector* выбрать *STM32F407VG*

STM32 Project

Target Selection
Select STM32 target

MCU/MPU Selector | Board Selector | Cross Selector

MCU/MPU Filters

Part Number Search: STM32F407VG

Core: >
Series: >
Line: >
Package: >
Other: >

Price = 5.623
IO = 82
Eeprom = 0 (Bytes)
Flash = 1024 (kBytes)
Ram = 192 (kBytes)
Freq. = 168 (MHz)

STM32F407VG

High-performance foundation line, ARM Cortex-M4 core with DSP and FPU, 1 Mbyte Flash, 168 MHz CPU, ART Accelerator, Ethernet, FSMC

ACTIVE Active
Product is in mass production

Unit Price for 10kU (US\$) : 5.623
Board: STM32F4DISCOVERY

LQFP100

The STM32F405xx and STM32F407xx family is based on the high-performance ARM® Cortex®-M4 32-bit RISC core operating at a frequency of up to 168 MHz. The Cortex-M4 core features a Floating point unit (FPU) single precision which supports all ARM single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security. The STM32F405xx and STM32F407xx family incorporates high-speed embedded memories (Flash memory up to 1 Mbyte, up to 192 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals.

MCUs/MPUs List: 1 item

* Part No.	Reference	Marketing Status	Unit Price for 10kU (U...	Board	Package	Flash	RAM	IO	Freq.	GFX Score
☆ STM32F407VG	STM32F407VGTX	Active	5.623	STM...	LQFP100	1024 kBytes	192 kBytes	82	168 MHz	0.0

< Back Next > Cancel Finish

Ввести имя проекта

STM32 Project

Project Setup
Setup STM32 project

Project Name:

Use default location

Location: Browse...

Options

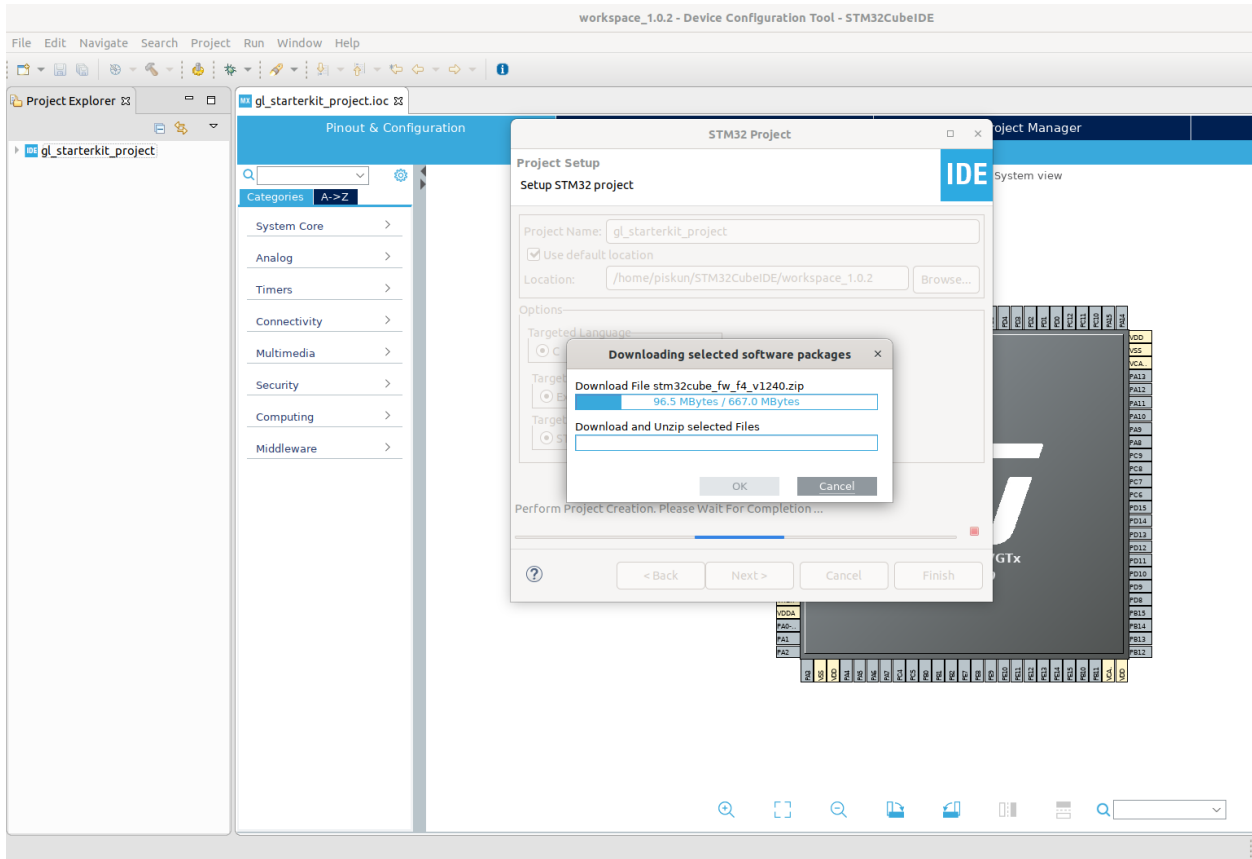
Targeted Language: C C++

Targeted Binary Type: Executable Static Library

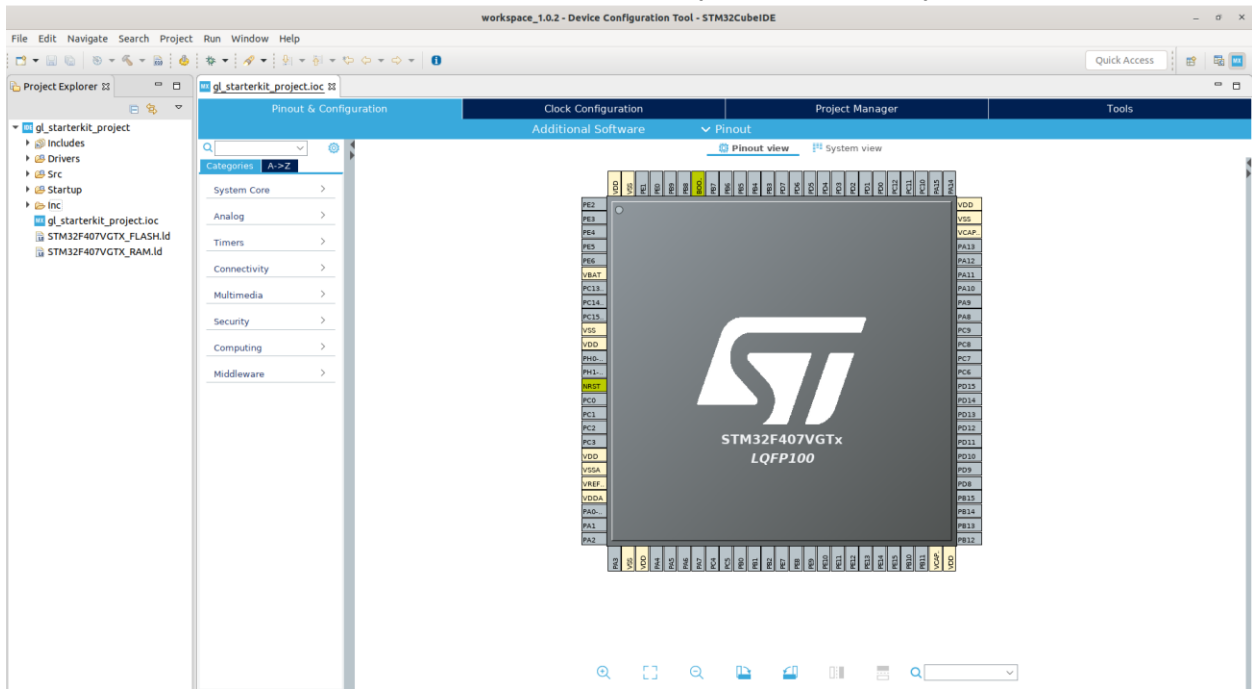
Targeted Project Type: STM32Cube Empty

? < Back Next > Cancel Finish

При создании первого проекта для микроконтроллера серии STM32F4 будет загружен Firmware Package (HAL, скрипты линкера, стартап файлы и т.п.).

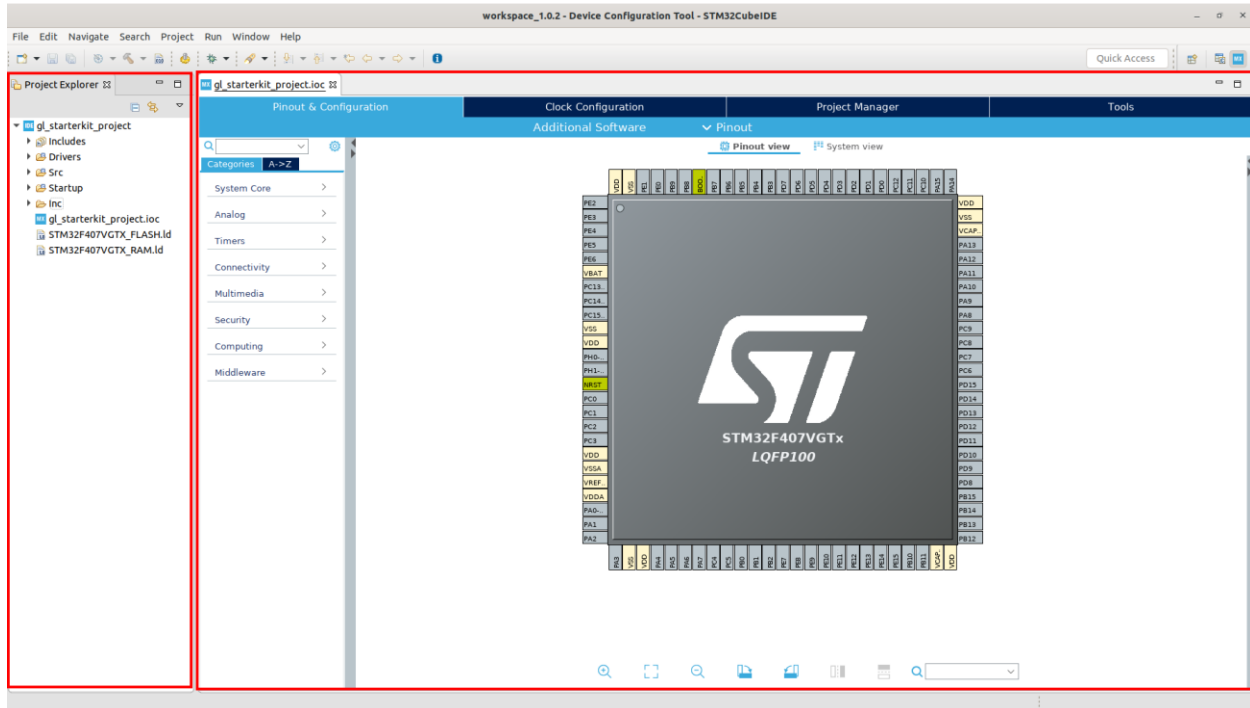


После завершения создания нового проекта IDE будет иметь следующий вид



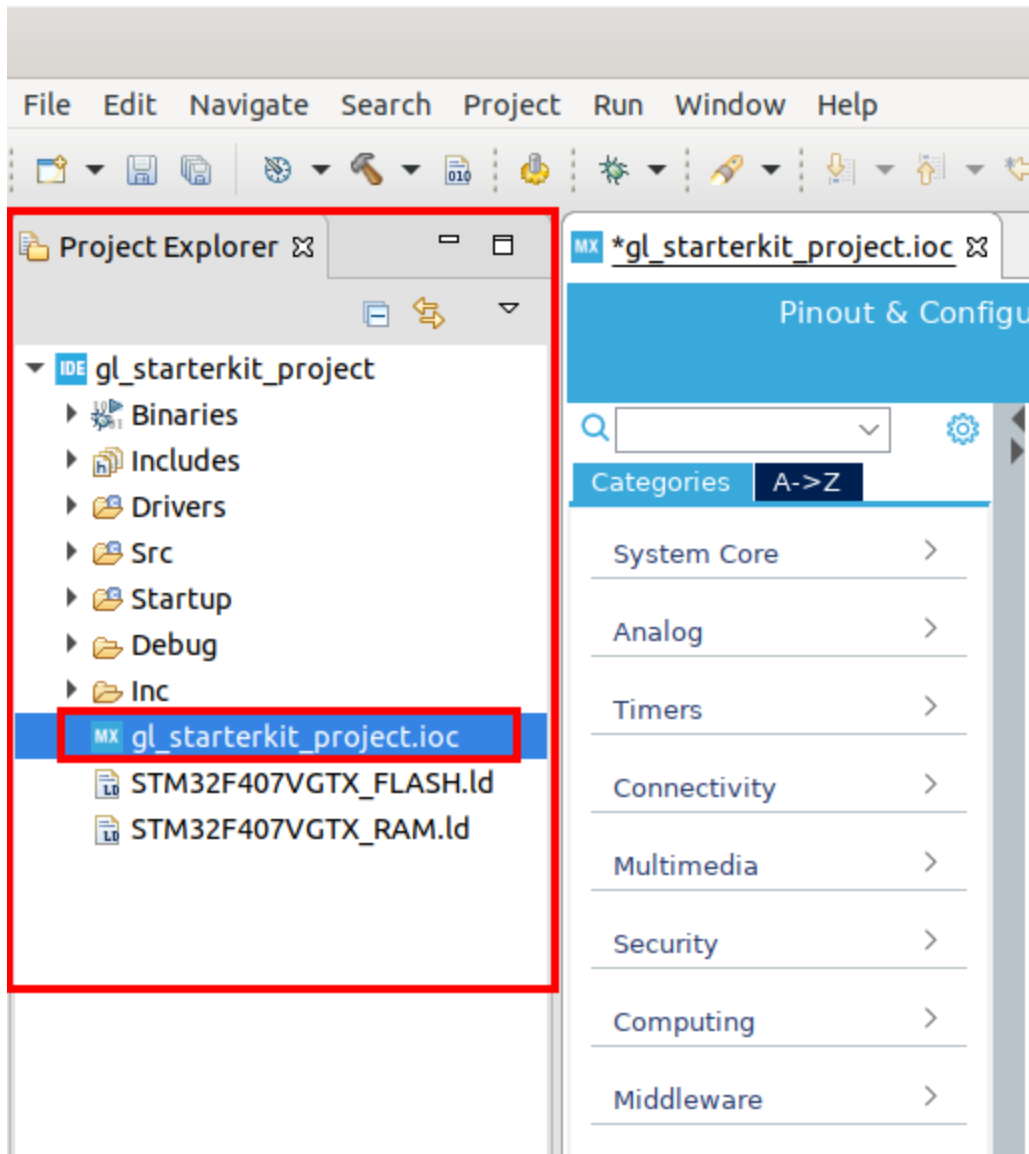
Главное окно IDE разделено на две части:

- *Project Explorer* (слева) - навигация по файлам проекта
- *Device Configuration Tool* (справа) - конфигурирование контроллера - тактирование, порты ввода/вывода, периферия, опции кодогенерации и т.д.



2. Обзор Device Configuration Tool

Для того, чтобы открыть окно Device Configuration Tool нужно дважды кликнуть по файлу с расширением *.ioc*



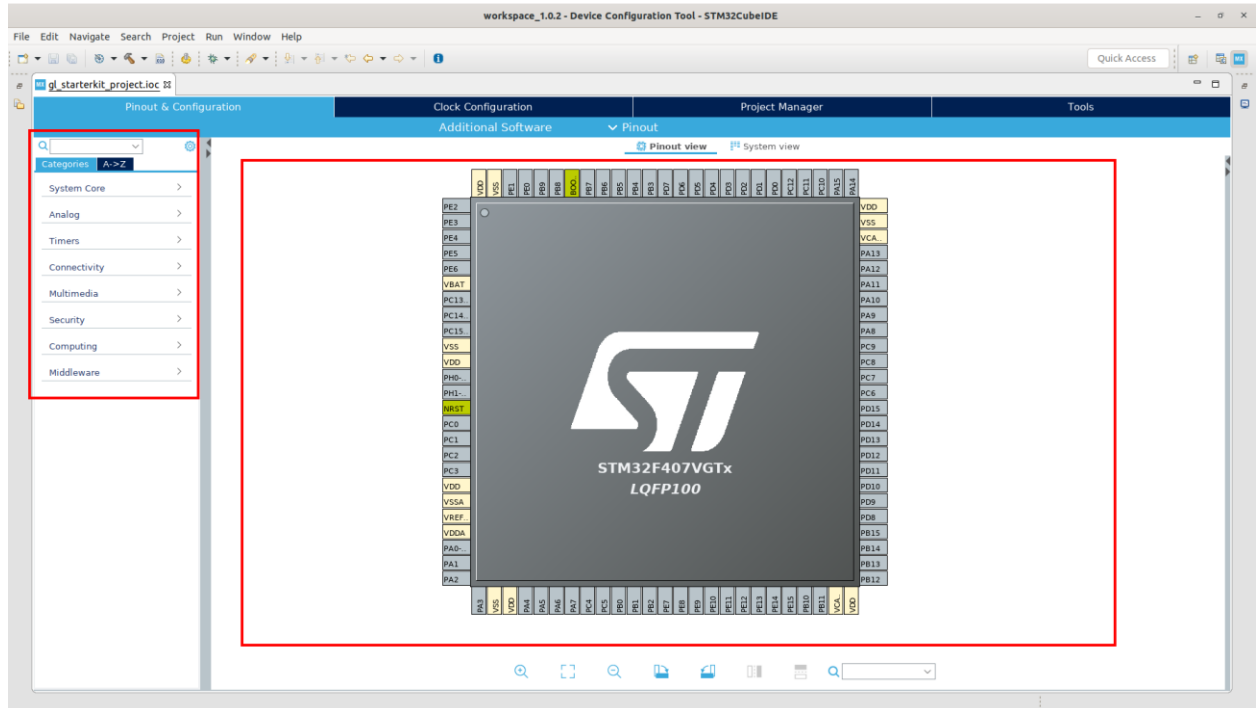
Device Configuration Tool состоит из четырех вкладок:

- *Pinout & Configuration* - настройка портов ввода/вывода и периферии;
- *Clock Configuration* - настройка тактирования - выбор тактового генератора, настройка PLL, частот шин, периферии;
- *Project Manager* - можно задать минимальный размер stack/heap, используемую версию Firmware Package, а также опции кодогенерации.
- *Tools* - можно приблизительно рассчитать потребление тока контроллером.

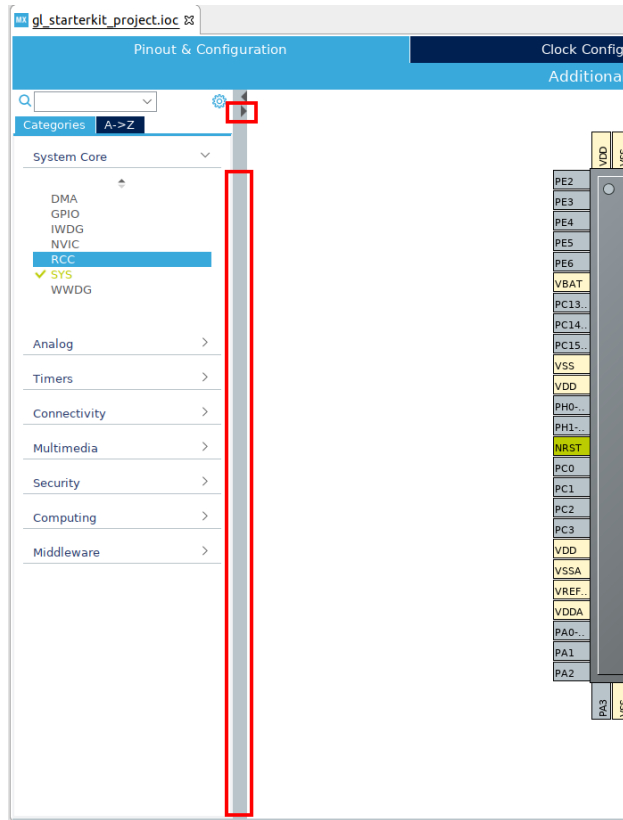
2.1 Pinout & Configuration

По умолчанию окно разделено на две части:

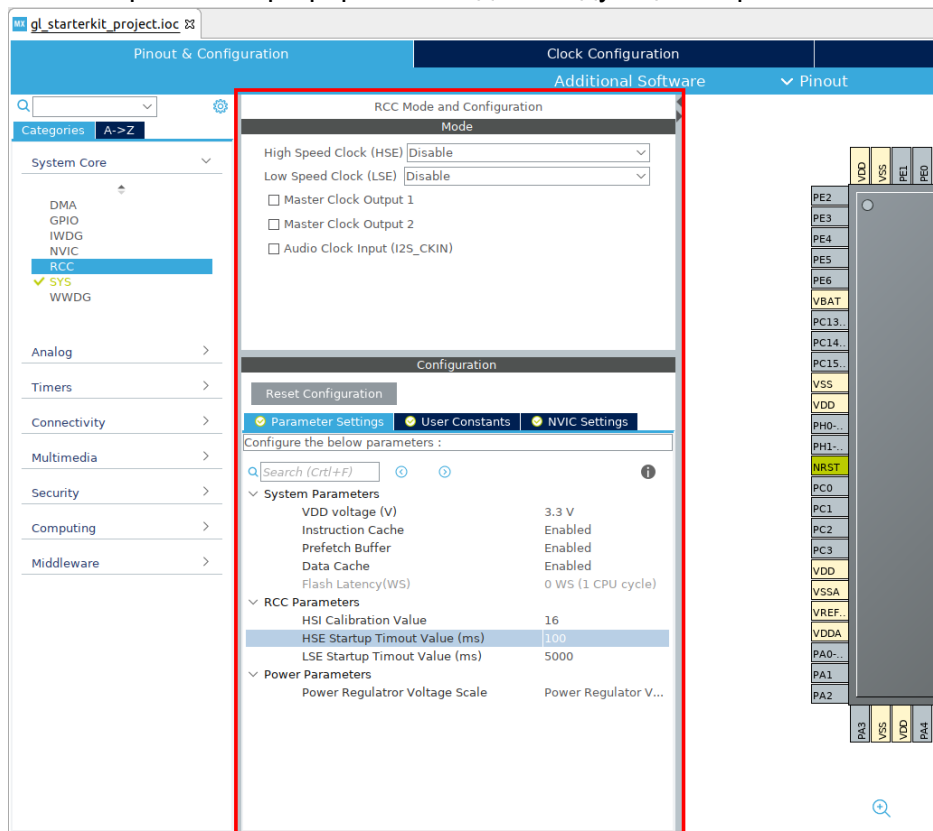
- *Peripherals Configuration* - периферия контроллера, сгруппированная по категориям (*Categories*). Также можно переключить список на отображение в алфавитном порядке (A->Z);
- *Pinout view* - отображение контроллера в выбранном корпусе (LQFP100) со всеми пинами (портами ввода/вывода).



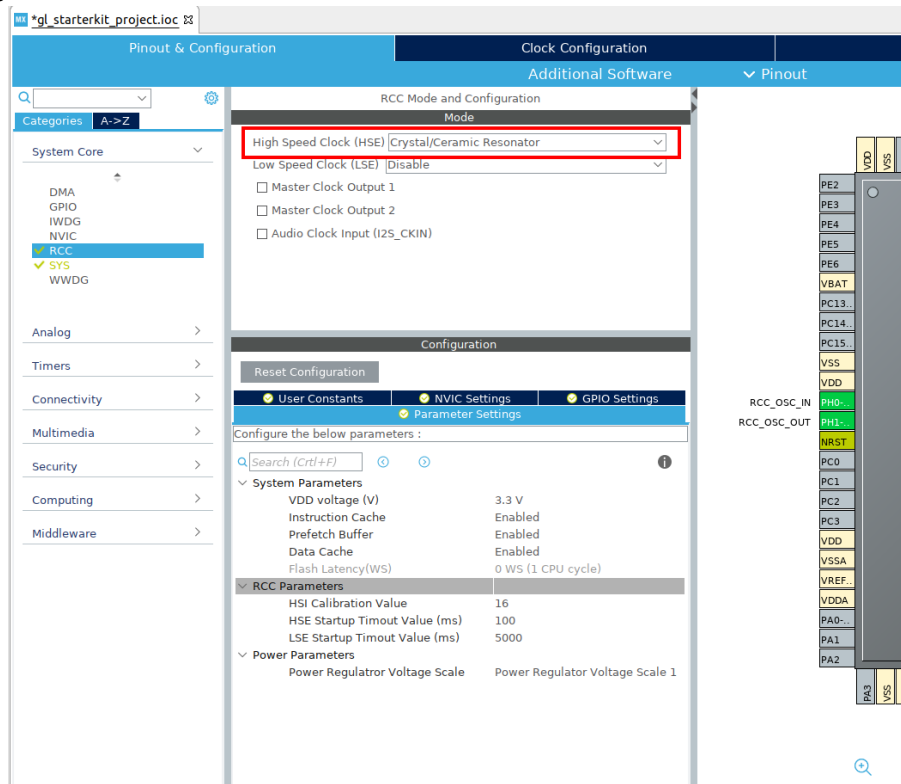
После включения/сброса MCU тактируется от внутреннего RC-генератора (HSI - High Speed Internal oscillator), который не отличается высокой стабильностью частоты. StarterKit имеет внешний кварц на 8 МГц. Переключим источник тактовых импульсов на HSE (High Speed External oscillator) - внешний кварцевый резонатор. В *Peripherals Configuration* выберем *System Core* -> *RCC*. По умолчанию окно настройки периферии скрыто. Для того, чтобы его открыть, нужно кликнуть на вертикальную полосу-разделитель между окнами *Peripherals Configuration* и *Pinout view* либо нажать на треугольник на той же полосе-разделителе



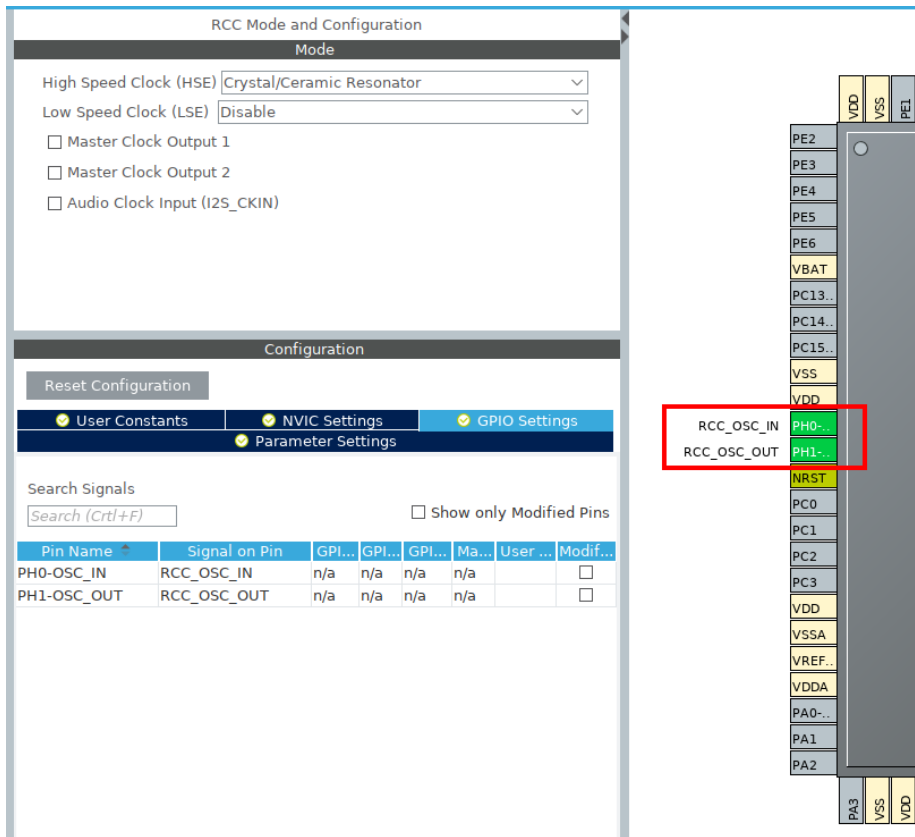
Окно настройки выбранной периферии выглядит следующим образом



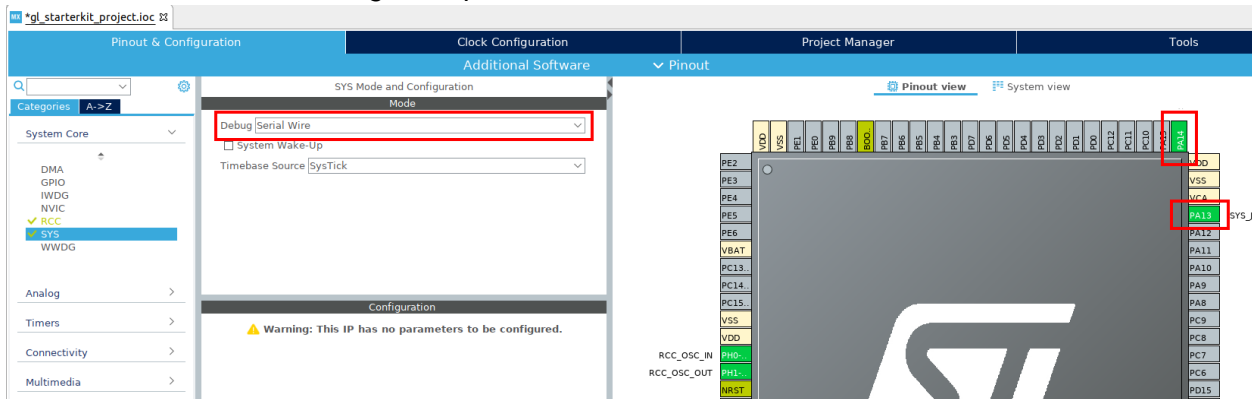
Включим тактирование от HSE. Для этого в выпадающем списке *High Speed Clock (HSE)* выберем *Crystal/Ceramic Resonator*



После этих действий в *Pinout View* пины PH0 и PH1 (OSC_IN и OSC_OUT соответственно) стали зелеными т.е. эти пины стали задействованы



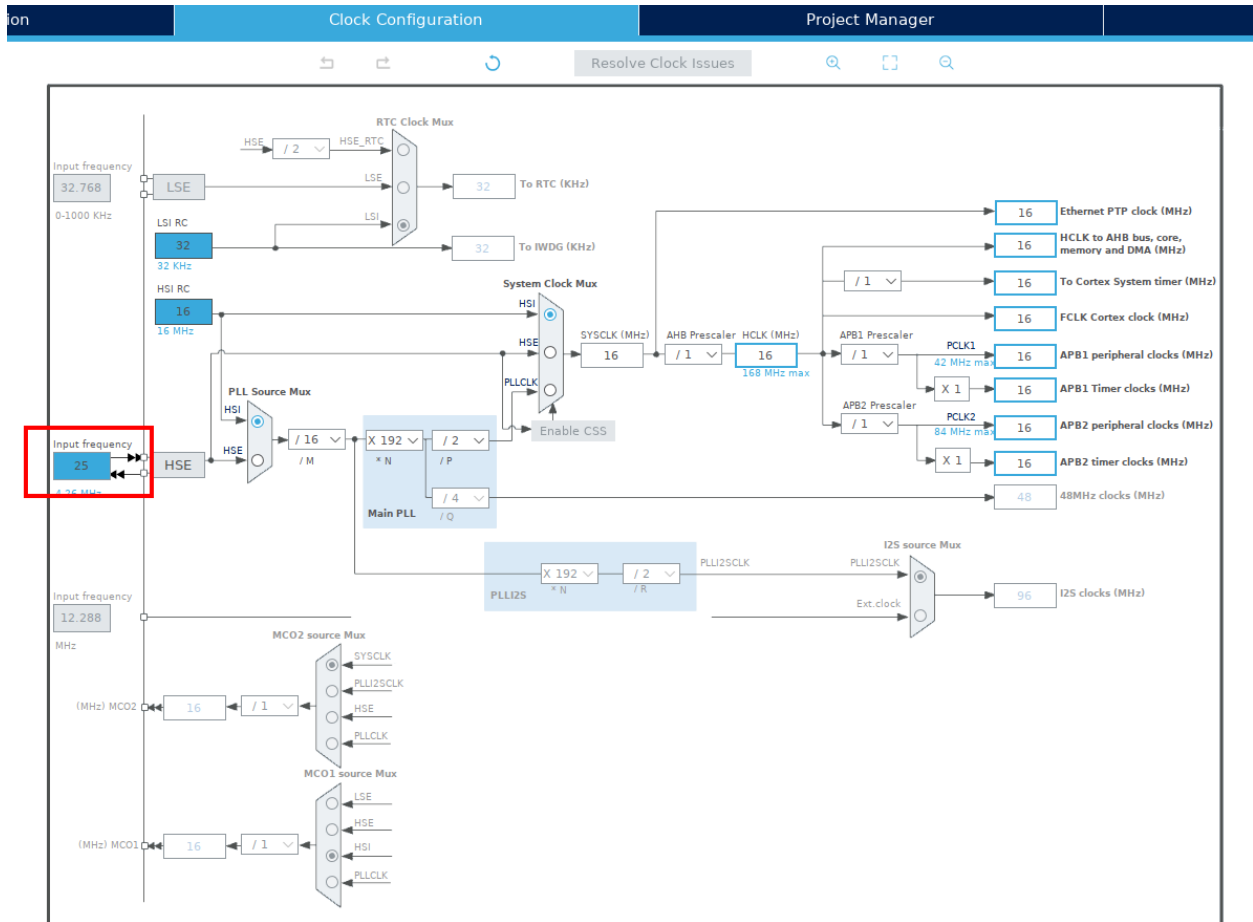
Далее необходимо разрешить отладку по SWD (последовательный отладочный интерфейс). Для этого в *Peripherals Configuration* переходим в категорию *SYS*. В выпадающем списке *Debug* выбираем *Serial Wire*



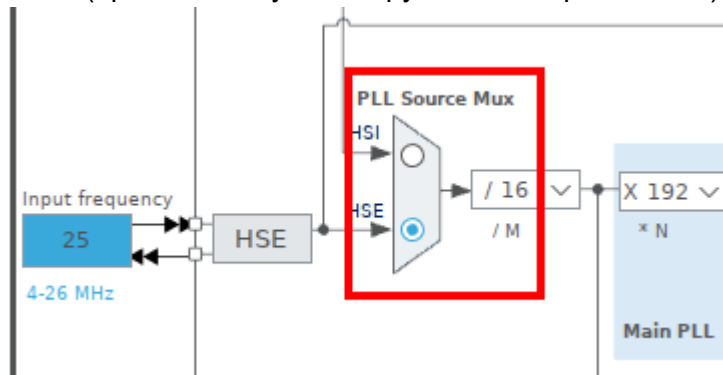
Так же видно, что в *Pinout View* пины PA13 (SWDIO) и PA14 (SWCLK) стали задействованы

3.1. Clock Configuration

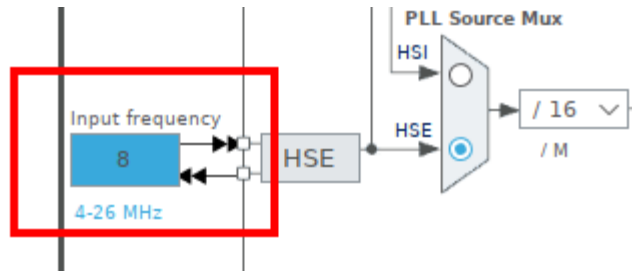
Показывает схему тактирования выбранного MCU



Так как мы включили тактирование от HSE, то блок *HSE* стал активным. Но источником тактовых импульсов по-прежнему является HSI. Для переключения на HSE в блоке *PLL Source Mux* выберем HSE (просто кликнуть на “кружочек” напротив HSE)



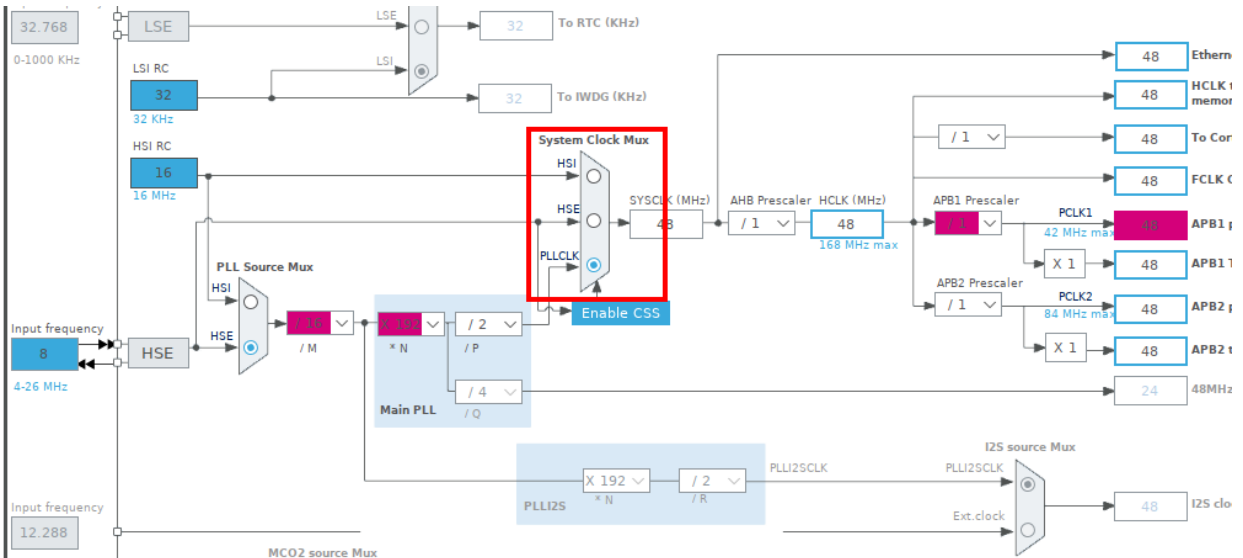
Как упоминалось ранее, на StarterKit стоит кварц на 8 МГц. Исправим *Input frequency*, т.к. по умолчанию стоит 25 МГц



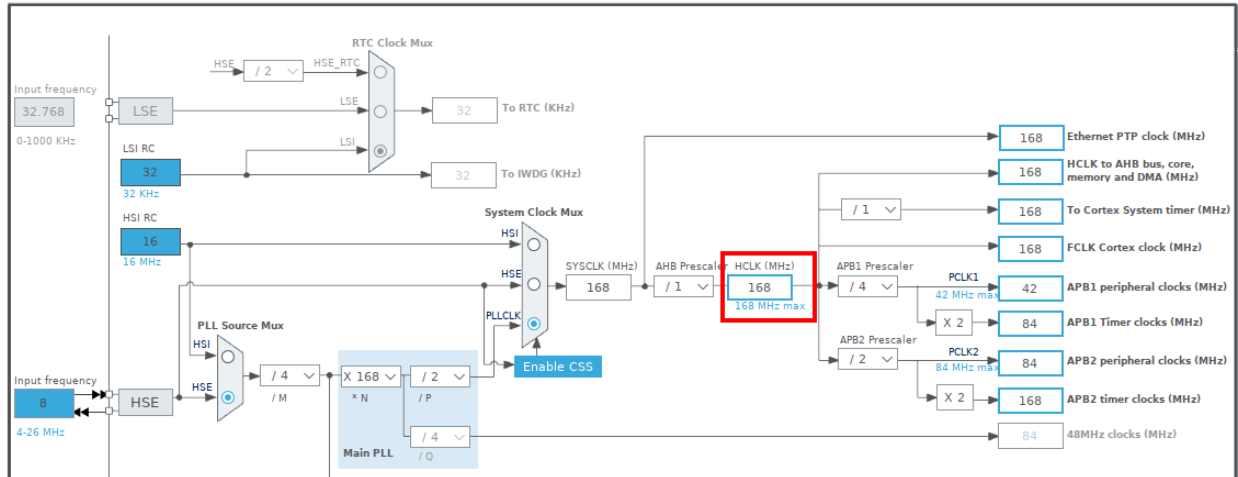
MCU имеет 3 шины:

- AHB - Advanced High-performance Bus. На этой шине работает ядро (core), DMA, память
- APB1/APB2 - Advanced Peripheral Bus. На этих шинах работают таймеры и почти вся периферия.

Максимальная частота AHB - 168 МГц. Для повышения частоты тактового генератора (HSE = 8МГц) задействуем PLL (Phase-Locked Loop). Для этого *System Clock Mux* выберем *PLLCLK*.



Красные значения делителей и частот означают, что частоты, после выбранных коэффициентов деления получились либо ниже допустимого предела, либо выше. Коэффициенты деления можно подобрать вручную. А можно эту задачу передать *Clock Configuration*. Так, например, если мы хотим получить частоту AHB = 168МГц, можно в *HCLK* ввести *168* и нажать *Enter*. Коэффициенты деления будут пересчитаны автоматически



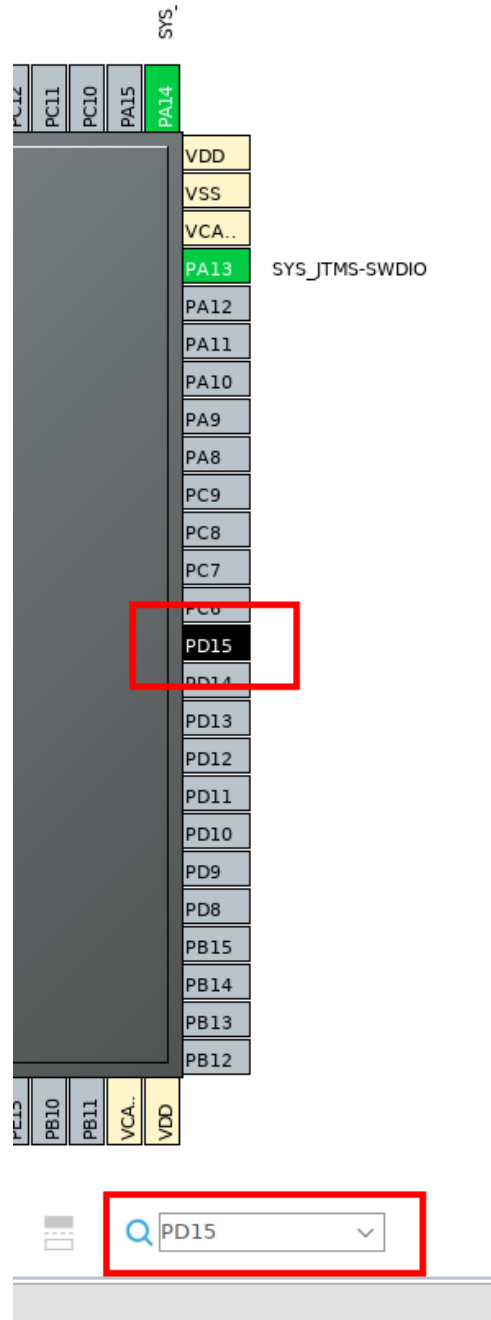
Настройка тактирования окончена. Теперь попробуем выполнить первую задачу - помигать светодиодом.

4. Мигание светодиодом

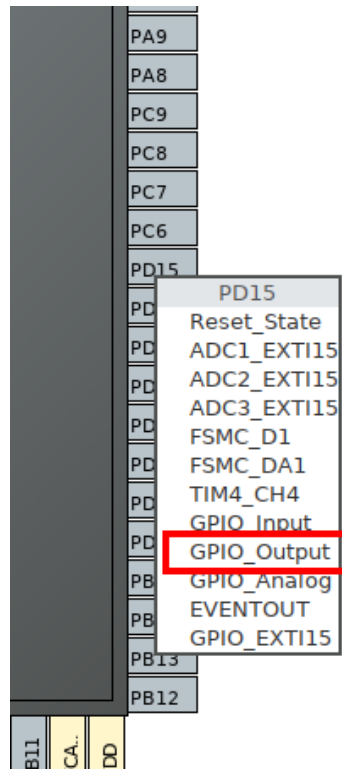
StarterKit имеет 4 светодиода:

- PD12 - зеленый;
- PD13 - оранжевый;
- PD14 - красный;
- PD15 - синий.

Помигаем, к примеру, синим светодиодом. Для этого нужно настроить пин #15 порта PORTD (PD15). Переходим на вкладку *Pinout & Configuration*. В *Pinout view* нужно найти пин PD15. Для этого можно воспользоваться поиском. В окне поиска введем *PD15*



Для управления светодиодом пин PD15 должен быть настроен на выход. Для этого в *Pinout view* кликнем на *PD15* и из списка выберем *GPIO_Output*



Далее переходим в *Peripheral Configuration* в категорию *System Core* в раздел *GPIO*. В списке появился PD15. После того, как мы выделим этот пин в списке, у нас появятся поля для настройки.

MX *gl_starterkit_project.ioc

Pinout & Configuration | Clock Configuration | Additional Software | Pinout

GPIO Mode and Configuration

Configuration

Group By Peripherals

GPIO RCC SYS

Search Signals
 Show only Modified Pins

Pin Name	Signal on Pin	GPIO output ...	GPIO mode	GPIO Pull-up/...	Maximum o...	User Label	Modified
PD15	n/a	Low	Output Push ...	No pull-up a...	Low		<input type="checkbox"/>

PD15 Configuration :

GPIO output level: Low

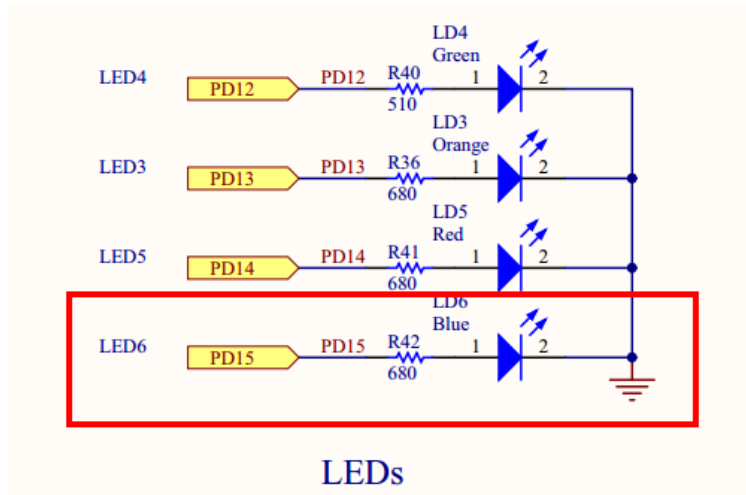
GPIO mode: Output Push Pull

GPIO Pull-up/Pull-down: No pull-up and no pull-down

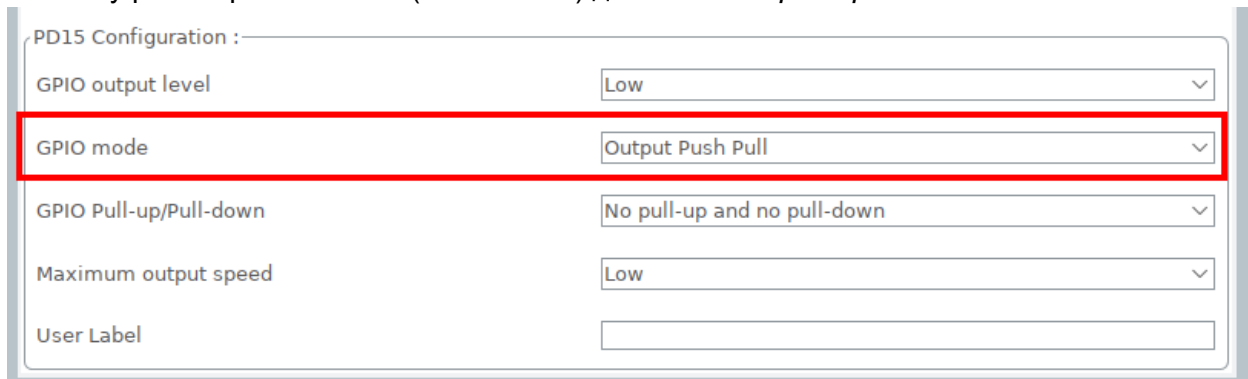
Maximum output speed: Low

User Label:

Посмотрим схему на STM32F4 Discovery board ([Discovery kit with STM32F407VG MCU](#)). По схеме синий светодиод, подключенный к PD15, не содержит подтягивающего резистора



Поэтому режим работы пина (GPIO mode) должен быть *push-pull*

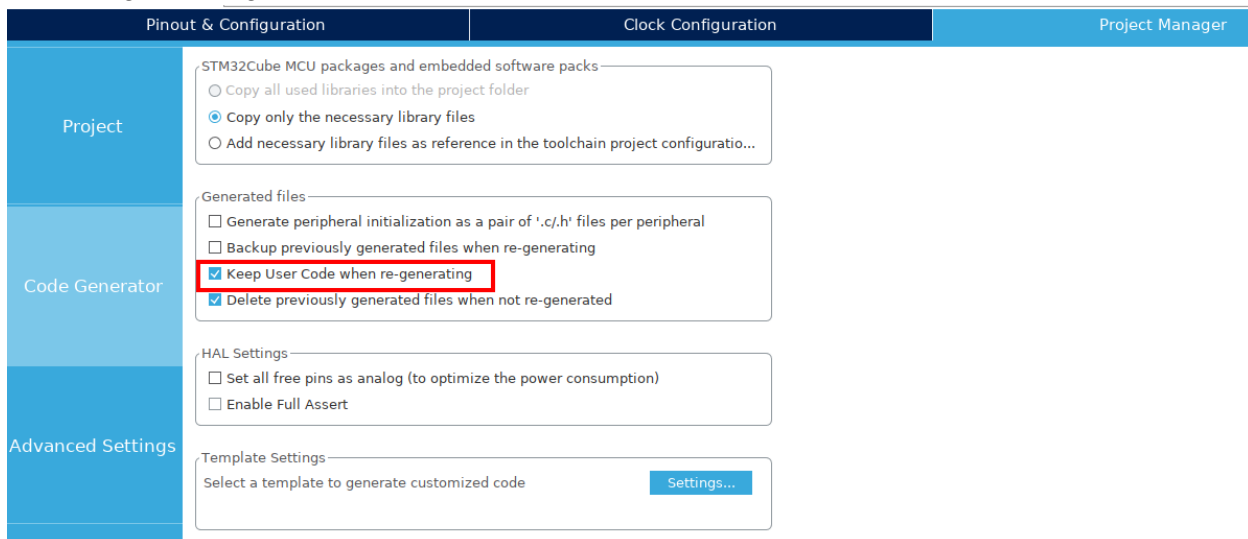


PD15 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	

Остальные настройки можно оставить без изменений.

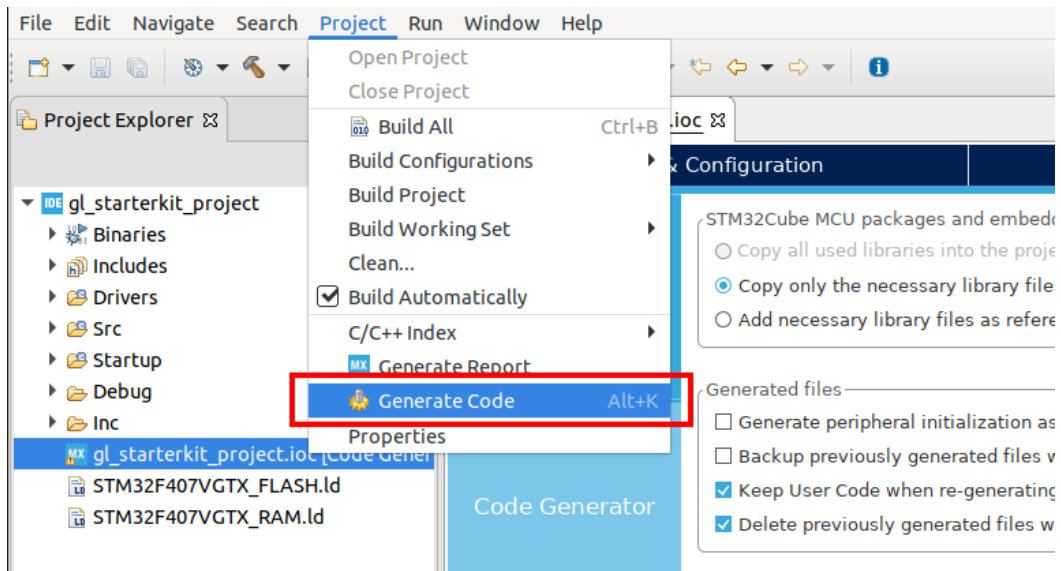
Теперь все готово для генерации кода. Для этого в *Device Configuration Tool* перейдем на вкладку *Project Manager* в *Code Generator* и поставим галочку напротив *Keep User Code when re-generating*



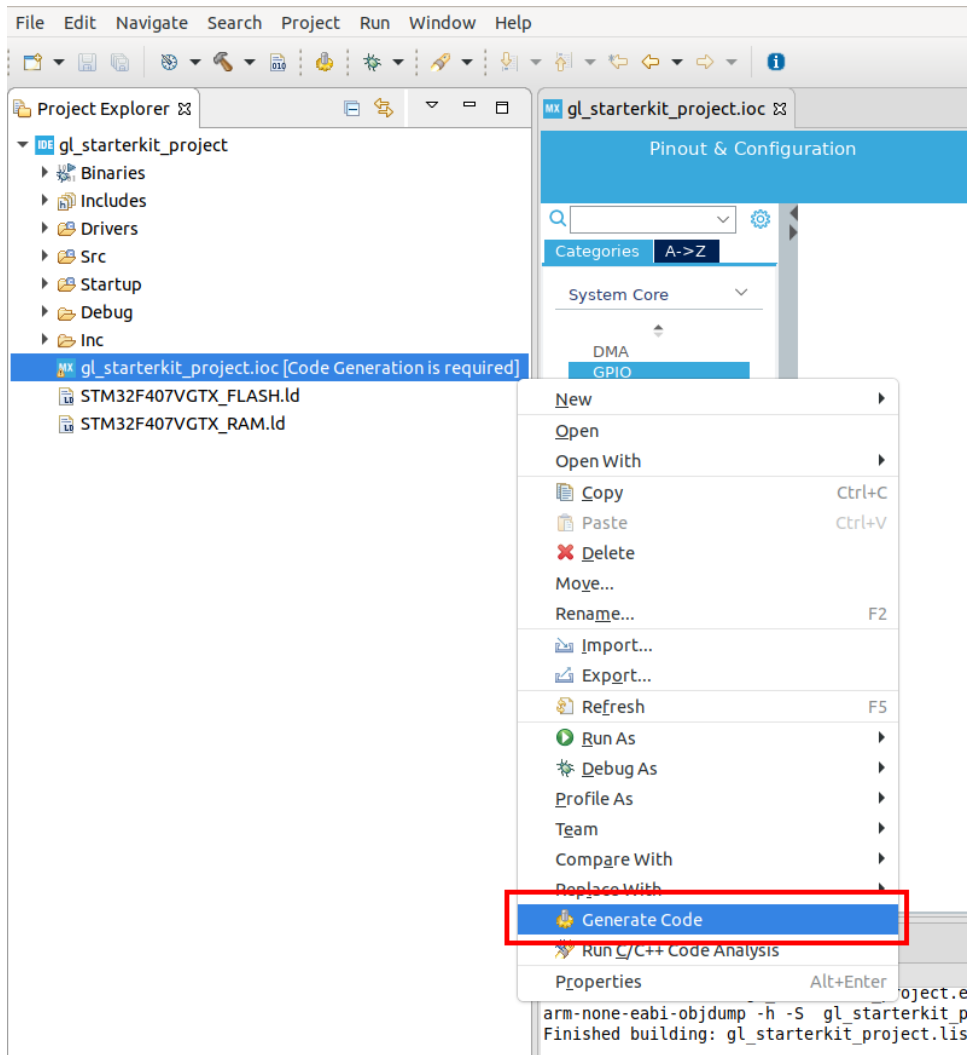
Pinout & Configuration	Clock Configuration	Project Manager
Project	STM32Cube MCU packages and embedded software packs <input type="radio"/> Copy all used libraries into the project folder <input checked="" type="radio"/> Copy only the necessary library files <input type="radio"/> Add necessary library files as reference in the toolchain project configuratio...	
Code Generator	Generated files <input type="checkbox"/> Generate peripheral initialization as a pair of '.c/.h' files per peripheral <input type="checkbox"/> Backup previously generated files when re-generating <input checked="" type="checkbox"/> Keep User Code when re-generating <input checked="" type="checkbox"/> Delete previously generated files when not re-generated	
Advanced Settings	HAL Settings <input type="checkbox"/> Set all free pins as analog (to optimize the power consumption) <input type="checkbox"/> Enable Full Assert	Template Settings Select a template to generate customized code <input type="button" value="Settings..."/>

Остальные настройки - по своему усмотрению.

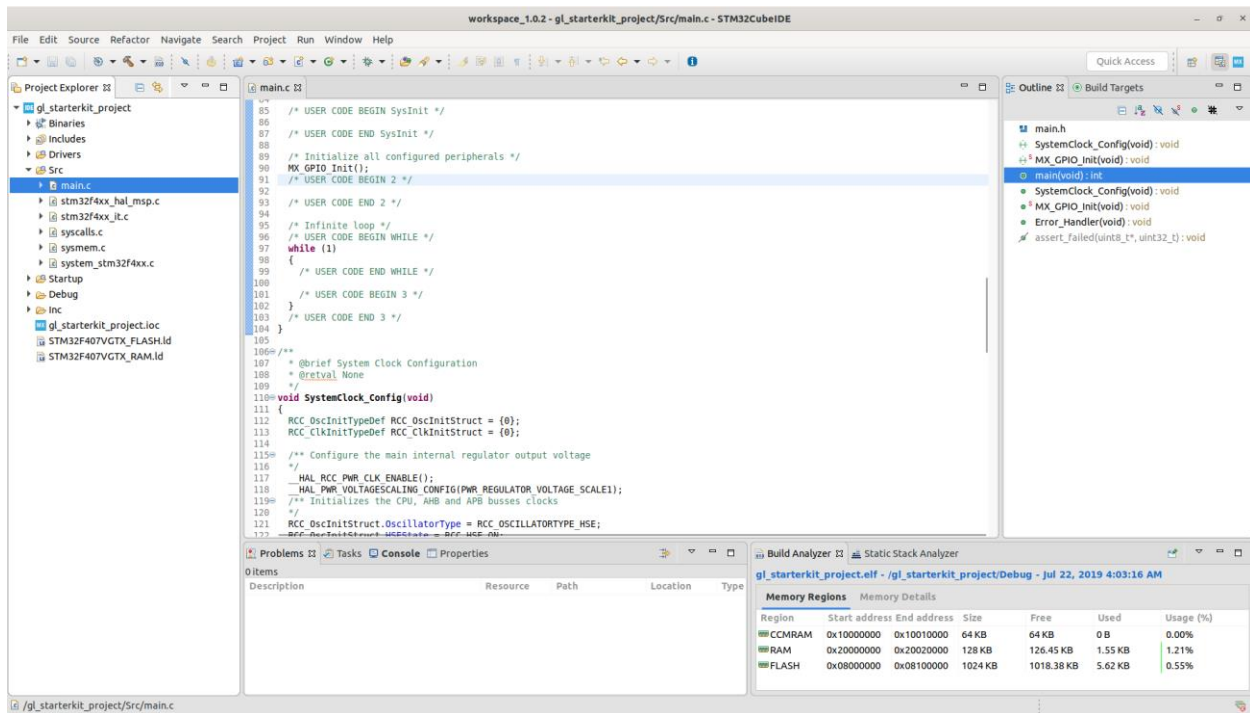
Для генерации кода перейти в *Project* -> *Generate Code (Alt+K)*



Либо в контекстном меню файла с расширением *.ioc* (*gl_starterkit_project.ioc*) выбрать *Generate Code*



В *Project Explorer* откроем файл *Src/main.c*



Инициализация PD15 выполняется в функции `static void MX_GPIO_Init(void)`

```

main.c main.h stm32f4xx_hal_conf.h stm32f4xx_hal_m
146 }
147
148 /**
149  * @brief GPIO Initialization Function
150  * @param None
151  * @retval None
152  */
153 static void MX_GPIO_Init(void)
154 {
155     GPIO_InitTypeDef GPIO_InitStruct = {0};
156
157     /* GPIO Ports Clock Enable */
158     __HAL_RCC_GPIOH_CLK_ENABLE();
159     __HAL_RCC_GPIOD_CLK_ENABLE();
160     __HAL_RCC_GPIOA_CLK_ENABLE();
161
162     /*Configure GPIO pin Output Level */
163     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
164
165     /*Configure GPIO pin : PD15 */
166     GPIO_InitStruct.Pin = GPIO_PIN_15;
167     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
168     GPIO_InitStruct.Pull = GPIO_NOPULL;
169     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
170     HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
171
172 }
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Теперь в цикле `while` добавим код для мигания светодиодом

```

89  /* Initialize all configured peripherals */
90  MX_GPIO_Init();
91  /* USER CODE BEGIN 2 */
92
93  /* USER CODE END 2 */
94
95  /* Infinite loop */
96  /* USER CODE BEGIN WHILE */
97  while (1)
98  {
99      /* Toggle PD15 output state */
100     HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
101     HAL_Delay(500);
102     /* USER CODE END WHILE */
103
104     /* USER CODE BEGIN 3 */
105 }
106 /* USER CODE END 3 */
107 }

```

```

HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
HAL_Delay(500);

```

ВАЖНО: для того, чтобы написанный код сохранялся после регенерации кода, он должен находиться между подобными комментариями

```
/* USER CODE BEGIN 2 */
```

```
/* USER CODE END 2 */
```

Собираем проект (*Project -> Build Project*). Если проект собрался успешно, то в *Console* должен быть следующий вывод

```

CDT Global Build Console
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_exti.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_flash.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_flash_ex.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_flash_ramfunc.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_gpio.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_pwr.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_pwr_ex.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_rcc.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_rcc_ex.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_tim.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc ..\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_tim_ex.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DU
arm-none-eabi-gcc -o "gl_starterkit_project.elf" @"objects.list" -mcpu=cortex-m4 -T"/home/piskun/STM32CubeIDE,
Finished building target: gl_starterkit_project.elf

arm-none-eabi-objdump -h -S gl_starterkit_project.elf > "gl_starterkit_project.list"
arm-none-eabi-size gl_starterkit_project.elf
text data bss dec hex filename
5736 20 1572 7328 1ca0 gl_starterkit_project.elf
Finished building: default.size.stdout

Finished building: gl_starterkit_project.list

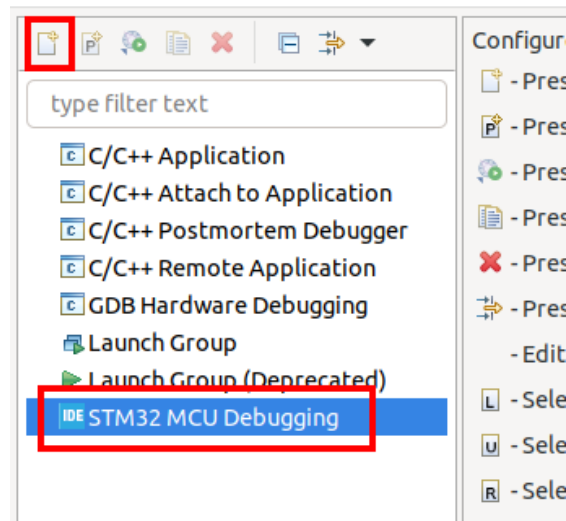
04:03:16 Build Finished. 0 errors, 0 warnings. (took 5s.116ms)

```

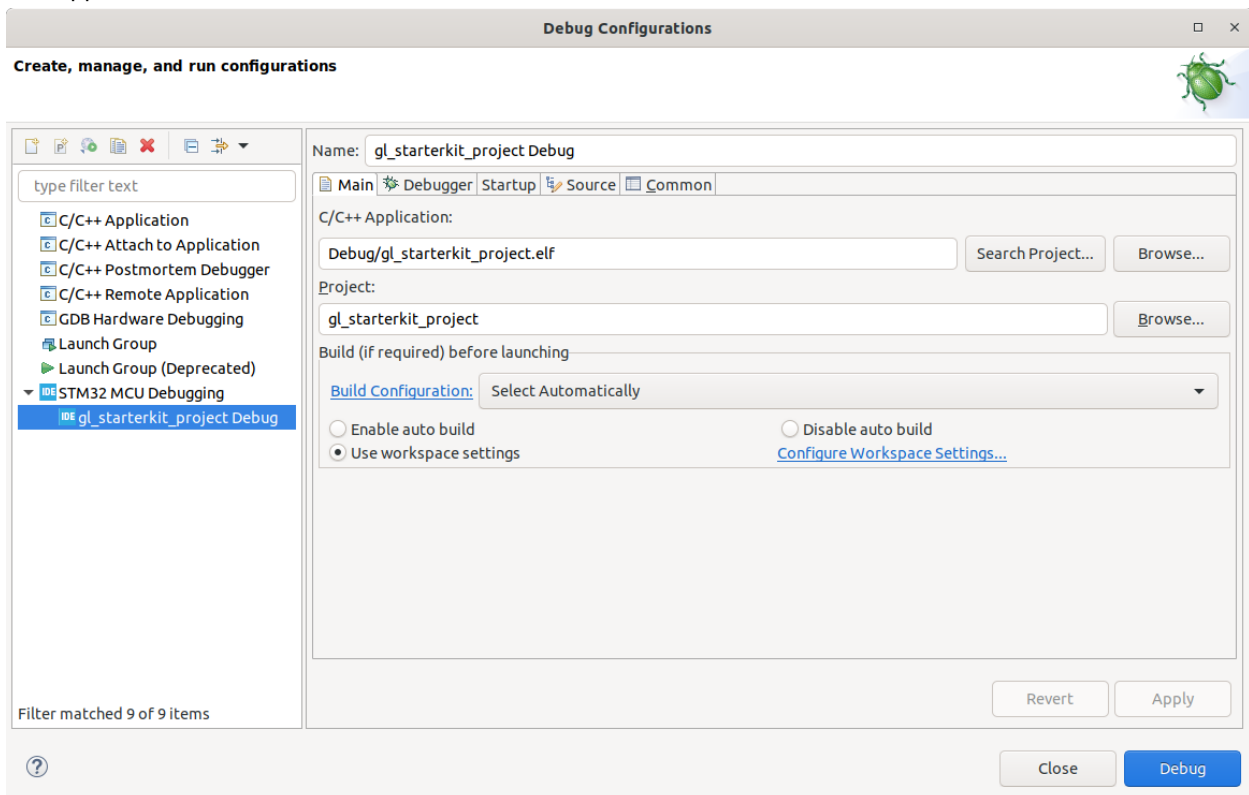
Создадим отладочную конфигурацию. Для этого перейдем в *Run -> Debug Configurations...*

В открывшемся окне выбрать *STM32 MCU Debugging* и нажать *New launch configuration*

Create, manage, and run configurations



Отладочная конфигурация должна быть настроена автоматически
Вкладка *Main*



Вкладка *Debugger*

Debug Configurations

Create, manage, and run configurations

Name: gl_starterkit_project Debug

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

GDB Connection Settings

- Autostart local GDB server Host name or IP address: localhost
- Connect to remote GDB server Port number: 61234

GDB Server Command Line Options

Interface

- SWD JTAG Use specific ST-LINK S/N

Serial Wire Viewer (SWV)

- Enable
- Clock Settings
 - Core Clock: 16.0 MHz
 - SWO Clock: 2000 kHz
- Port number: 61235
- Wait for sync packet

Misc

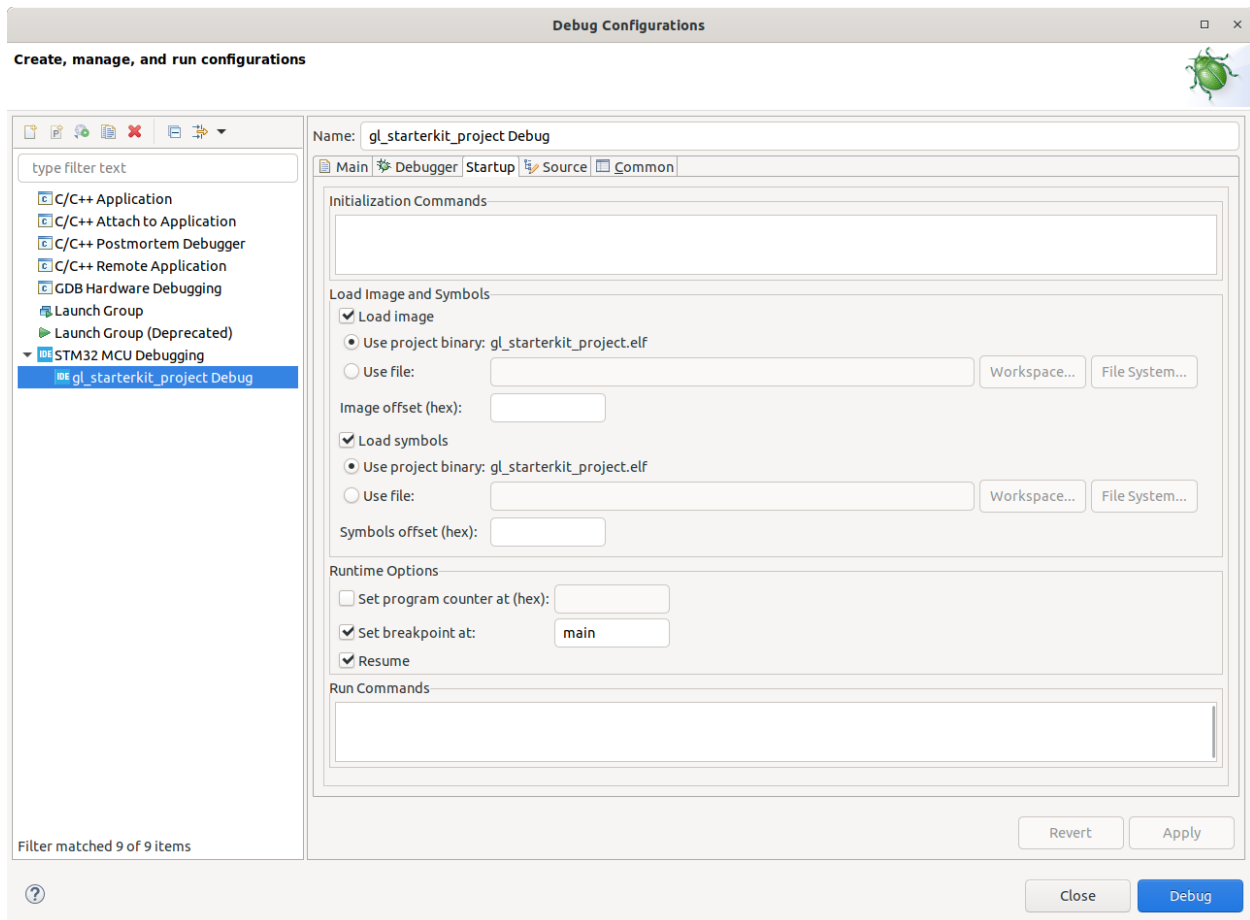
- Verify flash download
- Enable live expressions
- Log to file: /home/piskun/STM32CubeIDE/workspace_1.0.2/gl_starterkit_project/Debug/st-link_gdbs
- External Loader:
- Enable shared ST-LINK

Filter matched 9 of 9 items

Revert Apply

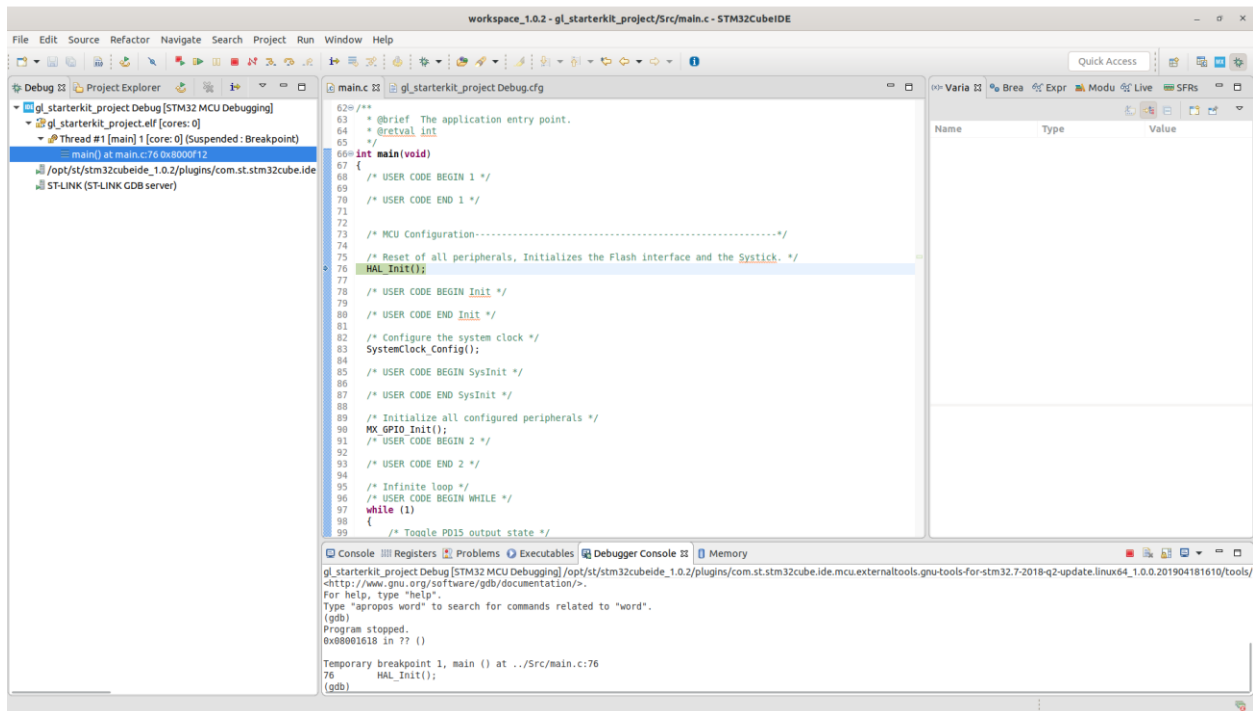
Close Debug

Вкладка *Startup*



Нажать кнопку *Debug*.

После старта отладочной сессии выполнение программы будет остановлено в начале функции *main()*.



F6/F5 - пошаговое выполнение программы;

F8 - продолжить выполнение. Выполнение будет остановлено на точке останова либо при нажатии кнопки “пауза” (*Suspend*).

5. Мигание светодиодом, используя таймер

Несмотря на то, что в предыдущем примере мы смогли помигать светодиодом, в нашей программе есть существенный недостаток - блокирующая задержка. Для того, чтобы частота мигания светодиода была различима глазу, в предыдущем примере мы добавили задержку в 500 мс после каждой смены состояния выхода *PD15*:

```
HAL_Delay(500);
```

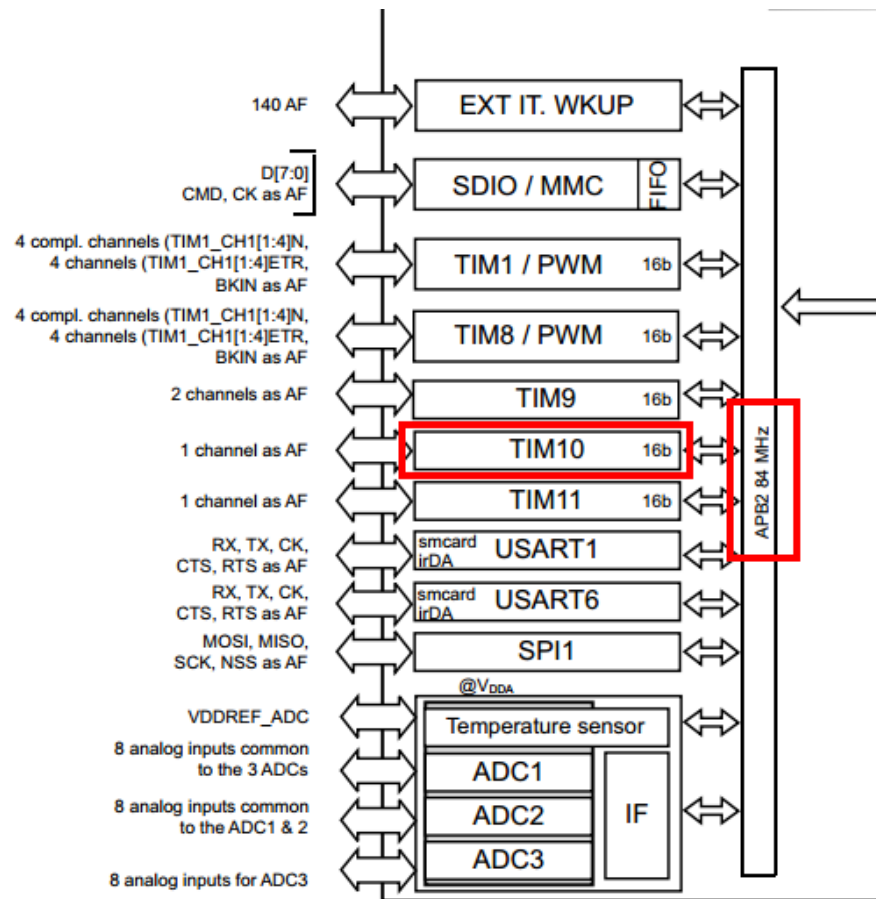
Если мы посмотрим на реализацию функции `__weak void HAL_Delay(uint32_t Delay)` мы увидим, что она является блокирующей:


```
main.c  gl_starterkit_project.ioc  stm32f4xx_hal.c  3
367  * @brief This function provides minimum delay (in milliseconds) based
368  *       on variable incremented.
369  * @note In the default implementation , SysTick timer is the source of
370  *       It is used to generate interrupts at regular time intervals whe
371  *       is incremented.
372  * @note This function is declared as __weak to be overwritten in case o
373  *       implementations in user file.
374  * @param Delay specifies the delay time length, in milliseconds.
375  * @retval None
376  */
377  weak void HAL_Delay(uint32_t Delay)
378  {
379  uint32_t tickstart = HAL_GetTick();
380  uint32_t wait = Delay;
381
382  /* Add a freq to guarantee minimum wait */
383  if (wait < HAL_MAX_DELAY)
384  {
385  wait += (uint32_t)(uwTickFreq);
386  }
387
388  while((HAL_GetTick() - tickstart) < wait)
389  {
390  }
391  }
392
```

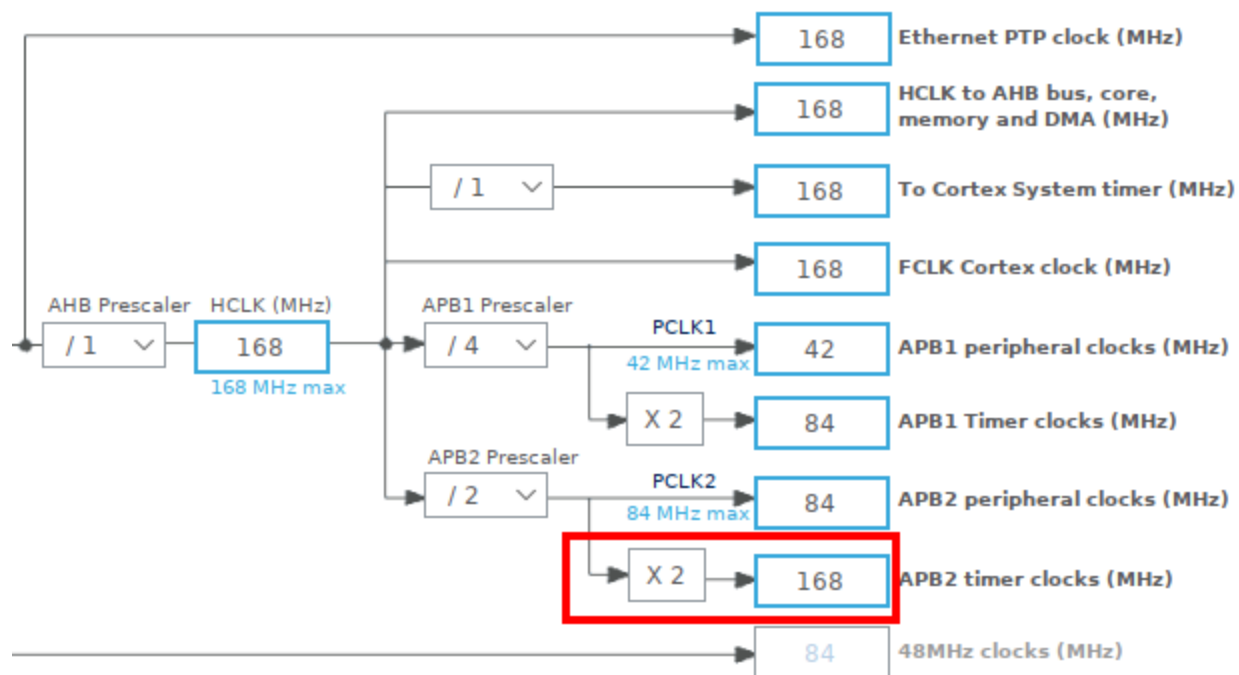
Все 500 мс контроллер только проверяет условие в цикле:

```
while((HAL_GetTick() - tickstart) < wait)
{
}
```

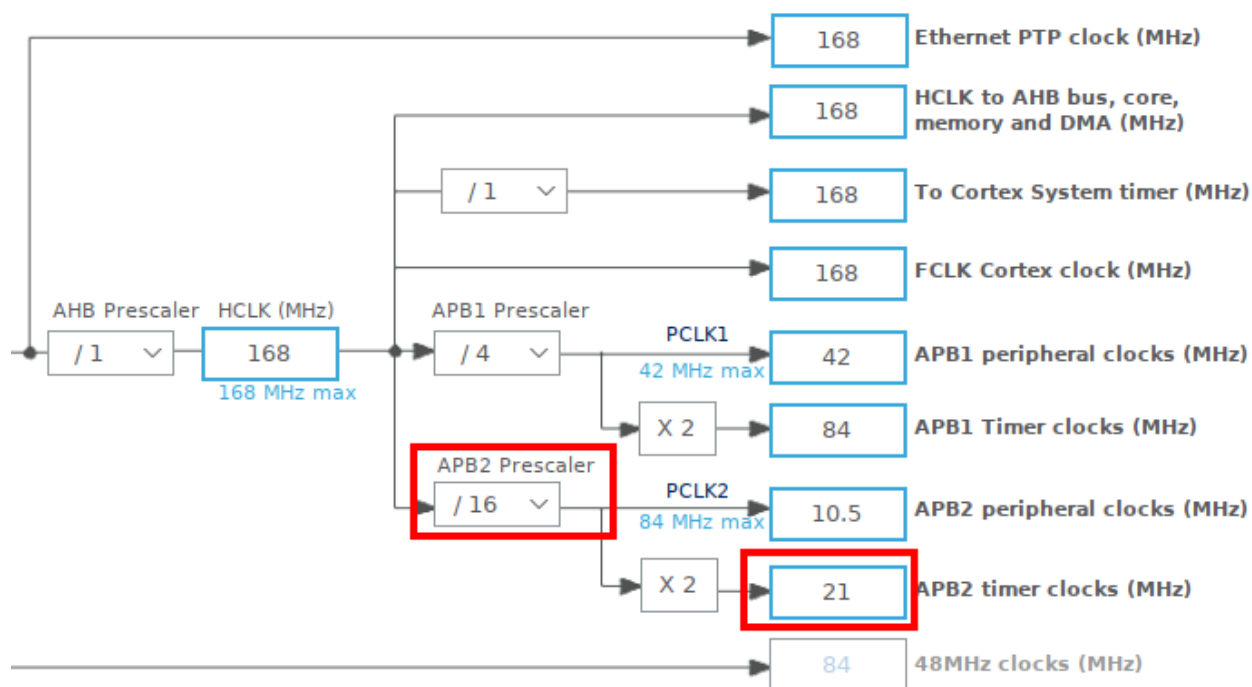
Исправим это. Сделаем так, чтобы состояние пина *PD15* менялось в прерывании таймера. Все остальное время контроллер может быть занят другими делами. Для нашей задачи вполне подойдет самый простой таймер - general-purpose timer - к примеру, *TIM10*. Для начала нужно определить к какой шине подключен *TIM10*. Для этого откроем datasheet (именно datasheet, а не reference manual), раздел *Device overview*, *STM32F40xxx block diagram*



Из диаграммы видно, что *TIM10* подключен к шине *APB2*. Перейдем на вкладку *Clock Configuration* в *STM32CubeIDE*. Частота шины *APB2* = 84 МГц, но частота тактирования таймеров шины *APB2* еще умножается на 2, поэтому таймеры этой шины будут работать на частоте 168 МГц

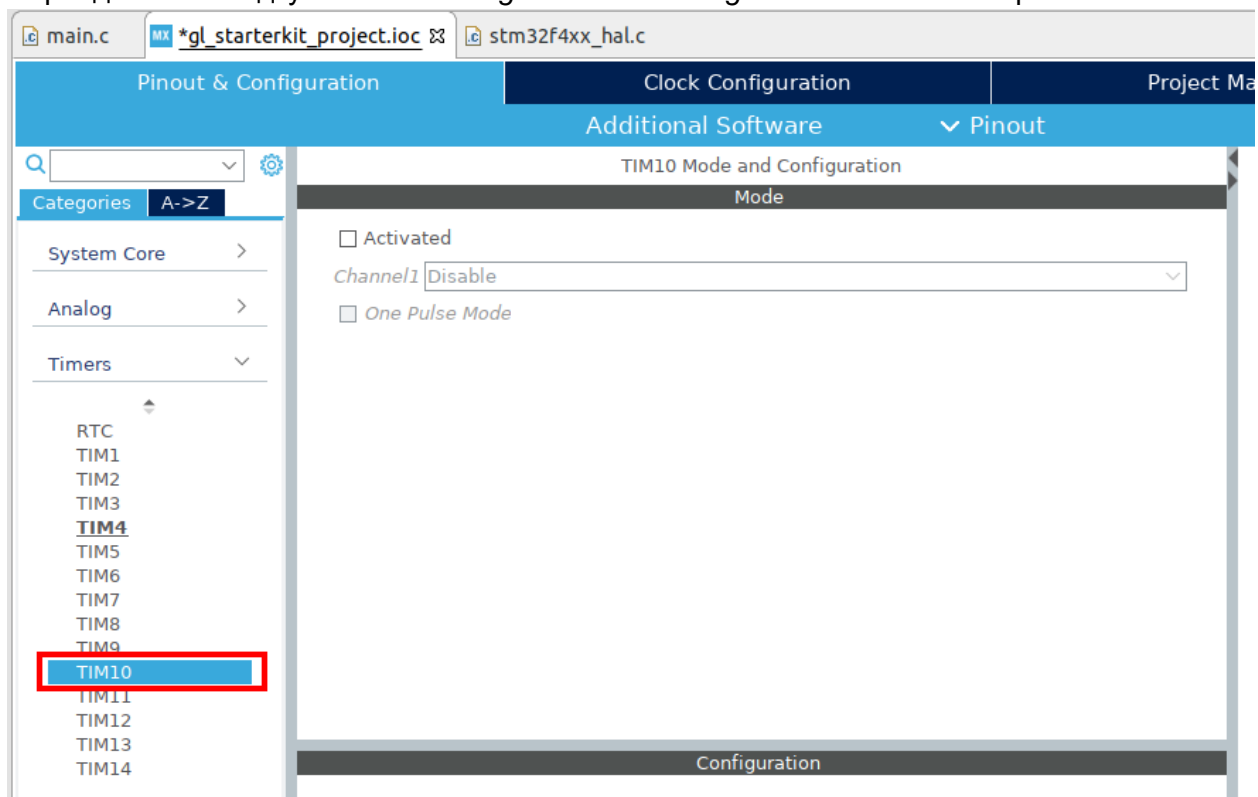


Получаем, что один тик $TIM10 = 1/168000000$ Гц = 0.00000000595 с = 0.00595 мкс. Мы хотим, чтобы пин $PD15$ менял состояние каждые 500 мс. Нет необходимости, чтобы $TIM10$ работал на такой большой частоте. К тому же, мы пока не используем никакую периферию, подключенную к шине $APB2$. Поэтому мы можем уменьшить частоту $APB2$ увеличив значение делителя (prescaler) этой шины. Установим максимально возможный делитель = 16

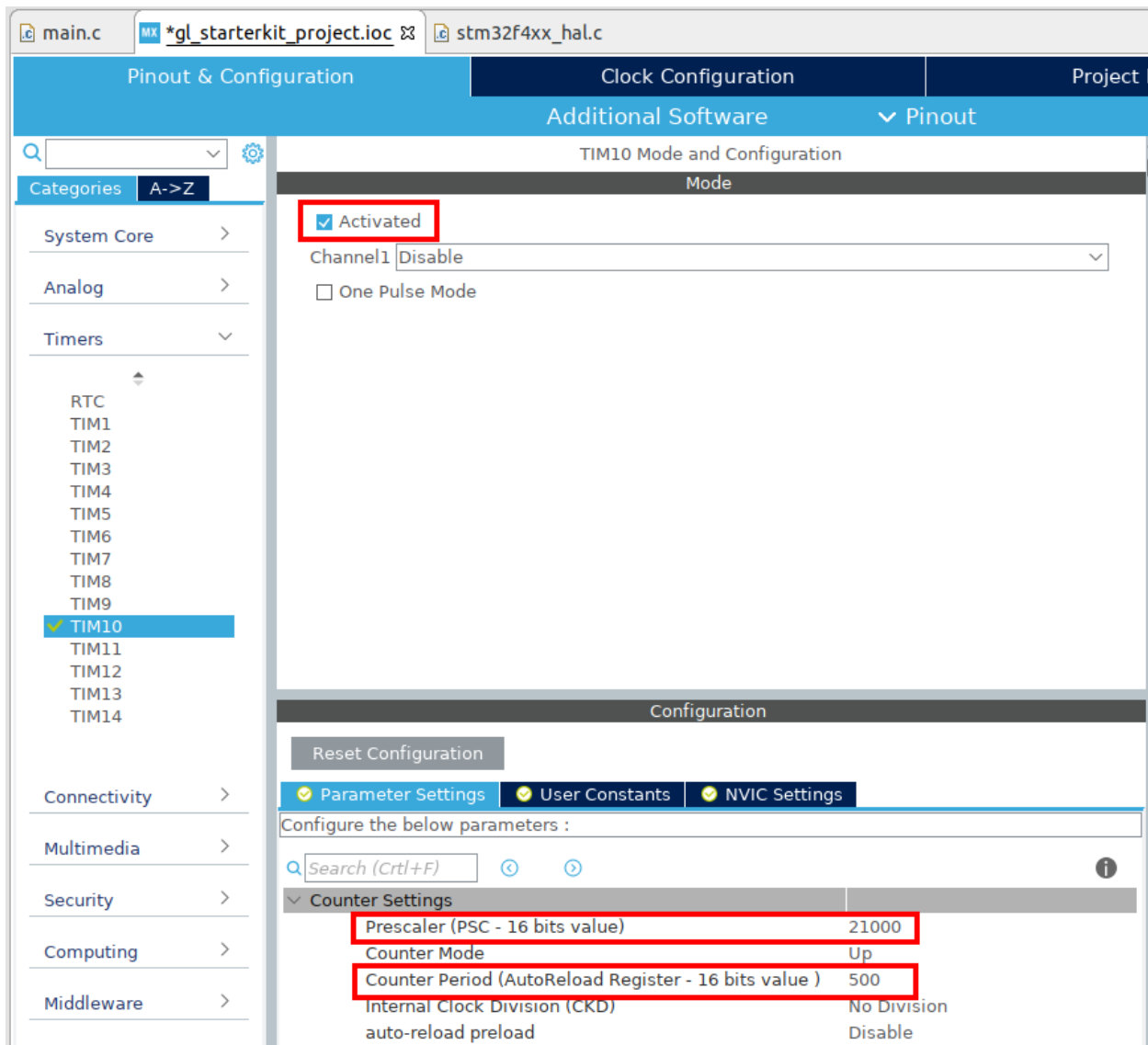


Теперь $TIM10$ будет работать на частоте 21 МГц. Следовательно, один тик $TIM10 = 1/21000000$ Гц = 0.0000000476 с = 0.0476 мкс.

Перейдем на вкладку *Pinout & Configurations*. В *Categories* -> *Timers* выберем *TIM10*



Настроим *TIM10* следующим образом



Мы задали делитель *Prescaler (PSC)* = 21000. Таким образом, *TIM10* будет работать на частоте: $21000000 \text{ Гц} / 21000 \text{ (PSC)} = 1000 \text{ Гц}$. Это значит, что счетчик таймера *TIM10* (регистр *TIM_CNT*) будет инкрементироваться с частотой 1000 Гц т.е. Каждую $1/1000 \text{ Гц} = 0.001 \text{ с} = 1 \text{ мс}$.

Counter Period (AutoReload Register) = 500. Это значит, что таймер будет инкрементировать свой счетчик (*TIM_CNT*) пока не достигнет значения 500. После этого таймер перезапустится и начнет счет с 0. Т.к таймер инкрементируется каждую 1 мс, то *TIM_CNT* достигнет значения 500 через: $1 \text{ мс} * 500 = 500 \text{ мс}$.

Теперь разрешим прерывание от *TIM10*. Для этого перейдем во вкладку *NVIC Settings* и разрешим *Update Interrupt*. Т.е. Прерывание будет срабатывать каждый раз, когда значение в регистре *TIM_CNT* будет становиться равным значению в регистре *AutoReload (ARR)* т.е. через каждые 500 мс.

The screenshot shows the STM32CubeMX IDE interface. The top tabs are 'Pinout & Configuration', 'Clock Configuration', and 'Project M'. Below these are 'Additional Software' and 'Pinout'. The main window is titled 'TIM10 Mode and Configuration'. On the left, a sidebar lists various components, with 'Timers' expanded to show a list from RTC to TIM14. 'TIM10' is selected and highlighted. The main area shows the 'Mode' section with 'Activated' checked, 'Channel1' set to 'Disable', and 'One Pulse Mode' unchecked. Below this is the 'Configuration' section, which includes a 'Reset Configuration' button and three tabs: 'Parameter Settings', 'User Constants', and 'NVIC Settings'. The 'NVIC Settings' tab is active and highlighted with a red box. Below the tabs is the 'NVIC Interrupt Table' with columns for 'Enabled', 'Preemption Prio...', and 'Sub Priority'. The row for 'TIM1 update interrupt and TIM10 global interrupt' has a checked box in the 'Enabled' column, also highlighted with a red box.

Enabled	Preemption Prio...	Sub Priority
<input checked="" type="checkbox"/>	0	0

Перегенерируем код.

Откроем файл *main.c*. У нас появилась функция инициализации таймера *TIM10*

```
main.c  gl_starterkit_project.ioc  stm32f4xx_it.c  stm32f4xx_hal_tim.c
150
151 /**
152  * @brief TIM10 Initialization Function
153  * @param None
154  * @retval None
155  */
156 static void MX_TIM10_Init(void)
157 {
158
159     /* USER CODE BEGIN TIM10_Init 0 */
160
161     /* USER CODE END TIM10_Init 0 */
162
163     /* USER CODE BEGIN TIM10_Init 1 */
164
165     /* USER CODE END TIM10_Init 1 */
166     htim10.Instance = TIM10;
167     htim10.Init.Prescaler = 21000;
168     htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
169     htim10.Init.Period = 500;
170     htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
171     htim10.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
172     if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
173     {
174         Error_Handler();
175     }
176     /* USER CODE BEGIN TIM10_Init 2 */
177
178     /* USER CODE END TIM10_Init 2 */
179
180 }
181
```

И вызов этой функции из *int main(void)*

```
main.c  gl_starterkit_project.ioc  stm32f4xx_it.c  stm32f4xx_hal_tim.c  startup_
66  * @retval int
67  */
68  int main(void)
69  {
70  /* USER CODE BEGIN 1 */
71
72  /* USER CODE END 1 */
73
74
75  /* MCU Configuration-----*/
76
77  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
78  HAL_Init();
79
80  /* USER CODE BEGIN Init */
81
82  /* USER CODE END Init */
83
84  /* Configure the system clock */
85  SystemClock_Config();
86
87  /* USER CODE BEGIN SysInit */
88
89  /* USER CODE END SysInit */
90
91  /* Initialize all configured peripherals */
92  MX_GPIO_Init();
93  MX_TIM10_Init();
94  /* USER CODE BEGIN 2 */
95
96  /* USER CODE END 2 */
97
98  /* Infinite loop */
99  /* USER CODE BEGIN WHILE */
100 while (1)
101 {
102  /* Toggle PD15 output state */
103  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
104  HAL_Delay(500);
105  /* USER CODE END WHILE */
106
107  /* USER CODE BEGIN 3 */
108  }
109  /* USER CODE END 3 */
110 }
```

Теперь перенесем вызов изменения состояния *PD15* из цикла *while* в обработчик прерывания *void TIM1_UP_TIM10_IRQHandler(void)* в файле *stm32f4xx_it.c*. Также удалим блокирующую задержку из цикла *while*. Таким образом наш главный цикл *while* остался пустым.

После инициализации *TIM10* (т.е. после вызова функции *MX_TIM10_Init()*) запустим *TIM10* с разрешенным прерыванием

```
HAL_TIM_Base_Start_IT(&htim10);
```



```
main.c  gl_starterkit_project.ioc  stm32f4xx_it.c  stm32f4xx_hal_tim.c  startup_
66  * @retval int
67  */
68  int main(void)
69  {
70  /* USER CODE BEGIN 1 */
71
72  /* USER CODE END 1 */
73
74
75  /* MCU Configuration-----*/
76
77  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
78  HAL_Init();
79
80  /* USER CODE BEGIN Init */
81
82  /* USER CODE END Init */
83
84  /* Configure the system clock */
85  SystemClock_Config();
86
87  /* USER CODE BEGIN SysInit */
88
89  /* USER CODE END SysInit */
90
91  /* Initialize all configured peripherals */
92  MX_GPIO_Init();
93  MX_TIM10_Init();
94  /* USER CODE BEGIN 2 */
95  HAL_TIM_Base_Start_IT(&htim10);
96  /* USER CODE END 2 */
97
98  /* Infinite loop */
99  /* USER CODE BEGIN WHILE */
100 while (1)
101 {
102     /* USER CODE END WHILE */
103
104     /* USER CODE BEGIN 3 */
105 }
106 /* USER CODE END 3 */
107 }
```

```
main.c  gl_starterkit_project.ioc  stm32f4xx_it.c  ⌵
168  */
169 void PendSV_Handler(void)
170 {
171     /* USER CODE BEGIN PendSV_IRQn 0 */
172
173     /* USER CODE END PendSV_IRQn 0 */
174     /* USER CODE BEGIN PendSV_IRQn 1 */
175
176     /* USER CODE END PendSV_IRQn 1 */
177 }
178
179 /**
180  * @brief This function handles System tick timer.
181  */
182 void SysTick_Handler(void)
183 {
184     /* USER CODE BEGIN SysTick_IRQn 0 */
185
186     /* USER CODE END SysTick_IRQn 0 */
187     HAL_IncTick();
188     /* USER CODE BEGIN SysTick_IRQn 1 */
189
190     /* USER CODE END SysTick_IRQn 1 */
191 }
192
193 /**
194  * STM32F4xx Peripheral Interrupt Handlers
195  * Add here the Interrupt Handlers for the used peripherals.
196  * For the available peripheral interrupt handler names,
197  * please refer to the startup file (startup_stm32f4xx.s).
198  */
199
200 /**
201  * @brief This function handles TIM1 update interrupt and TIM10 global interrupt.
202  */
203 void TIM1_UP_TIM10_IRQHandler(void)
204 {
205     /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */
206     /* Toggle PD15 output state */
207     HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
208     /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
209     HAL_TIM_IRQHandler(&htim10);
210     /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */
211
212     /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
213 }
214
```

Собираем проект и запускаем под отладкой

TODO: ADC