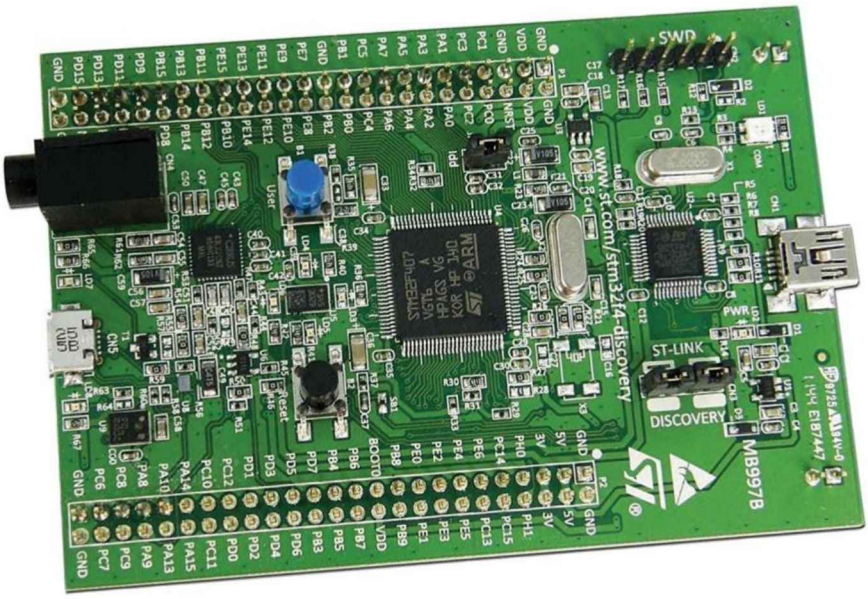
Pemrograman Sistem Embeded Berbasis ARM Cortex-M



Indra H Mulyadi
Eko Rudiawan
Anugerah Wibisana

POLIBATAM
PRESS

Pemrograman Sistem Embeded Berbasis ARM Cortex-M

Polibatam Press
Politeknik Negeri Batam

Pemrograman Sistem *Embedded* Berbasis ARM Cortex-M

Oleh :

Indra Hardian Mulyadi

Eko Rudiawan Jamzuri

Anugerah Wibisana

Polibatam Press
Politeknik Negeri Batam

Pemrograman Sistem *Embedded* Berbasis ARM Cortex-M

Penulis:
Indra Hardian Mulyadi
Eko Rudiawan Jamzuri
Anugerah Wibisana

Penyusunan Materi Pembelajaran ini dibiayai oleh:
Program Pengembangan Pendidikan Politeknik
Polytechnic Education *Development Project*
ADB LOAN 2928-INO
Direktorat Pembelajaran dan Kemahasiswaan – Ditjen Dikti
Kementerian Pendidikan dan Kebudayaan
Tahun Anggaran 2017

Perpustakaan Nasional: Katalog dalam terbitan
ISBN: 978-602-51056-4-7

Editor:	Indra Hardian Mulyadi
	Eko Rudiawan Jamzuri
	Anugerah Wibisana
Desainer:	Tegar Restiana
Ilustrator:	Mayang Marshita
Penata Letak:	Vella Nabillah
Pemeriksa Aksara:	Rivalta Luxyana Nur Hikmah

Penerbit:
Polibatam Press
Politeknik Negeri Batam
Jl. Ahmad Yani, Batam Center, Batam 29461

Prakata

Alhamdulillah, segala puji hanya untuk Allah SWT, Tuhan Yang Maha Memiliki ilmu, yang atas petunjukNya-lah kami dapat menyusun buku ajar ini. Diharapkan dengan adanya buku ini, mahasiswa dapat mengikuti perkuliahan dengan lebih baik dan dapat mengikuti latihan yang ada dalam buku ajar ini dengan lebih terstruktur.

Materi yang disusun pada buku ajar ini terdiri atas 13 bab dan disesuaikan dengan silabus mata kuliah Pemrograman Sistem Terbenam di program studi Teknik Mekatronika, Jurusan Teknik Elektro, Politeknik Negeri Batam. Mahasiswa diasumsikan sudah memperoleh dasar Pemrograman pada kuliah sebelumnya. Sejumlah materi ajar pada buku ajar ini akan disampaikan dalam satu semester. Isi buku ajar pada setiap topik bahasan disusun mulai dari teori penunjang setiap topik bahasan, praktikum, dan tugas membuat program untuk mengaplikasikan setiap topik bahasan, dan diakhiri dengan latihan soal. Dengan demikian, mahasiswa diharapkan lebih memahami setiap topik bahasan. Durasi aktivitas pada setiap bab (kecuali bab 1) terdiri atas 100 menit penyampaian teori ditambah 170 menit untuk praktikum dan tugas praktik. Latihan soal dapat dikerjakan oleh mahasiswa untuk memperkuat pemahaman terhadap teori dan praktikum yang telah dilaksanakan.

Kit Praktikum yang digunakan dalam buku ini adalah STM32F407G-DISC1, sebuah *Development kit* berbasis mikrokontroler STM32F407VGT6 (ARM Cortex-M4). Pemrograman dilakukan dengan menggunakan Keil µVision dan STM32CubeMX sehingga sangat memudahkan dalam melakukan pemrograman.

Meskipun buku ajar ini telah selesai dihadirkan dan siap digunakan, akan tetapi masih tidak menutup adanya kekurangan pada setiap topik bahasannya. Segala masukan, kritik dan saran sangat kami harapkan untuk semakin menyempurnakannya pada kesempatan mendatang.

Batam, 24 November 2017

Penyusun

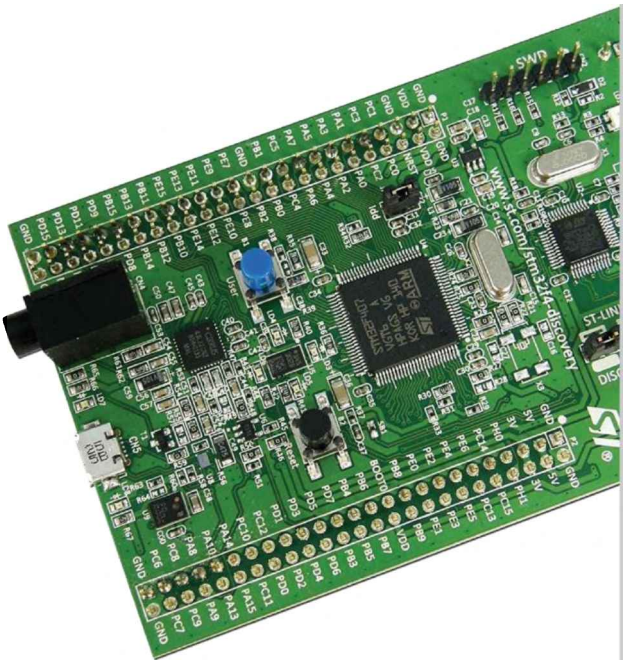
Daftar Isi

Daftar Gambar

Daftar Tabel

Sistem Embeded Berbasis ARM Cortex-M

1



Bab 1 Sistem *Embedded* dan ARM Cortex M4

Bab ini membahas mengenai sistem *embedded* (sistem terbenam), mikrokontroler, ARM, dan mikrokontroler yang digunakan pada praktikum di buku ini, yakni STM32F407VGT6.

1.1. Sistem *Embedded*

> *"An Embedded system is an Application- specific computer system which is built into a larger system or device."* (Conrad & Dean, 2011)

Saat ini, kita menjumpai sistem komputer di semua bidang kehidupan, baik sistem komputer yang skala besar sampai skala yang kecil. Berbeda dengan *General Purpose computer* (seperti *Personal Computer* (PC)), Sistem *embedded* digunakan untuk aplikasi dengan tujuan yang spesifik yang biasanya memiliki batasan *Real Time*. Contoh sistem *embedded* yang dapat kita jumpai sehari-hari adalah kamera *digital*, mesin cuci, *microwave*, kalkulator, telepon genggam, jam tangan, *printer*, *scanner*, mesin faksimili, dan masih banyak lagi. Angka penjualan mikroprosesor di dunia untuk sistem *embedded* - yakni sekitar 98% - jauh melampaui angka penjualan mikroprosesor untuk komputer.

Sistem *embedded* memiliki beberapa karakteristik, di antaranya:

1) Fungsi-tunggal, yaitu hanya mengeksekusi satu jenis program secara berulang-ulang. Tidak seperti PC yang mengeksekusi beberapa macam program
2) Sangat dibatasi dari segi harga, ukuran, daya, dan performansi.
3) *Real Time*, yakni eksekusi tanpa *delay*[1]

1.2. Mikrokontroler

Secara sederhana, mikrokontroler dapat didefinisikan sebagai komputer kecil di dalam sebuah *Integrated Circuit* (IC). Sistem

[1] Sebenarnya setiap aplikasi memiliki *delay*, tidak betul-betul *Real Time*. Batasan *delay* maksimum pada aplikasi *Real Time* didefinisikan sendiri oleh penggunanya, tidak ada definisi mutlak.

embedded tidak bisa dilepaskan dengan mikrokontroler. Ketika kita menyebut mikroprosesor dan mikrokontroler, keduanya terkadang dianggap sama, namun pada dasarnya memiliki perbedaan. Mikroprosesor terdiri atas *Arithmetic Logic Unit* (ALU), *instruction decoder*, sejumlah *register*, dan *input/Output digital*. Beberapa mikroprosesor memiliki tempat penyimpanan sementara yang disebut *cache*. Berbeda dengan mikroprosesor, mikrokontroler merupakan sebuah sirkit kecil dalam satu *chip* tunggal yang mengandung semua komponen dari sebuah sistem komputer, yaitu mikroprosesor itu sendiri, ditambah dengan memori dan *peripheral* (seperti *Analog to Digital Converter* (ADC), *Digital to Analog Converter* (DAC), *Pulse Width Modulation* (PWM), dan sebagainya).

Sebuah mikrokontroler memiliki komponen utama sebagai berikut:

1) Central Processing Unit (CPU)
2) Memori
3) Periferal
4) Sistem *clock* (Osilator)

Ada banyak perusahaan pembuat mikrokontroler, di antaranya Atmel, Analog Devices, Dallas Semiconductor, Freescale, Infineon, Intel, Fujitsu, Micro*chip*, National Semiconductor, NXP, NEC, Renesas, Cypress, Samsung, Texas Instruments, ST Microelectronics, Zilog, dan Nuvoton. IC Insight mengeluarkan data pada 2016 mengenai data penjualan mikrokontroler di dunia yang dapat dilihat pada Tabel 1.

Tabel 1. Suplier MCU

2016 Rank	Company	2015	2016	% Change	% Marketshare
1	NXP*	1,350	2,914	116%	19%
2	Renesas	2,560	2,458	-4%	16%
3	Microchip**	1,355	2,027	50%	14%
4	Samsung	2,170	1,866	-14%	12%
5	ST	1,514	1,573	4%	10%
6	Infineon	1,060	1,106	4%	7%
7	Texas Instruments	820	835	2%	6%
8	Cypress***	540	622	15%	4%

*Acquired Freescale in December 2015.

**Purchased Atmel in April 2016.

***Includes full year of sales from Spansion acquisition in March 2015.

Source: IC Insights, company reports

MCU yang digunakan pada buku ini merupakan MCU 32-bit. Dilihat dari tren penjualan, MCU 32-bit terus naik grafiknya. Kini, pengguna MCU 4-bit, 8-bit, dan 16-bit semakin banyak beralih ke MCU 32-bit. Alasan utamanya adalah dengan harga yang relatif sama, diperoleh performansi yang jauh lebih tinggi.

Bit, Byte, dan Word

Bit adalah sebuah digit biner, yakni bernilai 0 atau 1. Bit ini dapat diimplementasikan dengan menggunakan saklar (ON atau OFF). Byte adalah urutan bit-bit sebanyak 8 buah, sehingga bernilai antara 00000000 s.d 11111111 (0 s.d 255). Word adalah sejumlah bit-bit yang dianggap satu unit oleh CPU. Lebar dari word ini tergantung dari jenis CPU. Beberapa CPU menggunakan 32-bit word, dan ada juga menggunakan 16-bit word ataupun 64-*bit* word. Data diambil dari memori ke prosesor dalam potongan word dan dimanipulasi oleh ALU dalam potongan word. Semakin besar ukuran word, maka pemrosesan data akan semakin cepat dan fleksibel. Dalam aplikasi yang digunakan pada buku ini, kita gunakan definisi pada Tabel 2.

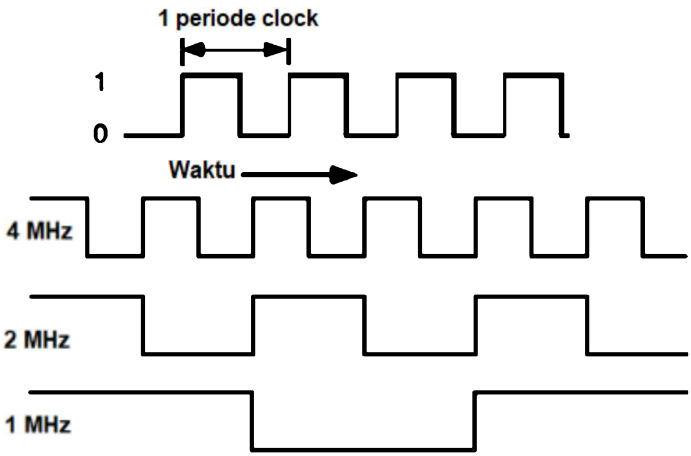
Tabel 2. Definisi byte, word, half-word, dan double word pada buku ini

Istilah	ukuran bit
byte	8-bit

word	32-bit
half-word	16-bit
double-word	64-bit

Clock

Mikrokontroler memiliki sebuah sistem pengaturan waktu yang dinamakan *clock*, yang berdetik (*tick*) dengan frekuensi yang teratur (Gambar 1). Untuk mengeksekusi sebuah instruksi, mikrokontroler memerlukan satu atau beberapa *clock*, tergantung jenis mikrokontrolernya. Sebagai contoh, mikrokontroler jenis AVR yang memiliki *crystal* 16 MHz memiliki kecepatan eksekusi 16 *Mega Instructions Per Second* (MIPS), sedangkan jenis mikrokontroler PIC hanya memiliki kecepatan 4 MIPS dengan *crystal* yang sama. Ini dikarenakan mikrokontroler PIC memerlukan empat *clock* untuk mengeksekusi sebuah instruksi. Sumber *clock* dapat berasal dari *crystal*, osilator RC baik internal maupun eksternal, dan *clock* eksternal.



Gambar 1. Diagram waktu *Clock*

Program *Memory*

Program dari sebuah mikrokontroler disimpan dalam memori *flash* yang bersifat permanen (*non volatile*), yang tidak akan terhapus

walaupun suplai daya dimatikan. Memori *flash* ini terdiri atas dua bagian. Bagian pertama digunakan untuk menyimpan program aplikasi, sedangkan bagian kedua digunakan untuk *booting*, yaitu program yang dieksekusi segera setelah suplai daya diaktifkan.

Data *Memory*

Data dapat disimpan dalam *Random Access Memory* (RAM) yang bersifat sementara (*volatile*), yakni akan terhapus ketika suplai daya dimatikan. Dalam pemrograman C, sebuah variabel biasanya disimpan dalam RAM ini. Jika ingin menyimpan data secara permanen, pengguna dapat menggunakan *Read-Only Memory* (ROM).

Register

Register merupakan alat penyimpanan kecil yang mempunyai kecepatan akses cukup tinggi, yang digunakan untuk menyimpan data dan instruksi yang sedang diproses. Secara umum, *register* dibagi menjadi dua jenis, yakni:

1) General Purpose *Register* (GPR)
2) Special Function *Register* (SFR)

GPR berfungsi untuk menyimpan data sementara, misalnya untuk operasi aritmatika/logika pada ALU. Sedangkan SFR memiliki fungsi khusus yang telah ditentukan oleh produsennya. Karena nilai dari bit-bitnya terhubung langsung secara fisik dengan sirkit di dalam MCU, maka perubahan bit-bit ini akan langsung berpengaruh pada operasi MCU. Sebagai contoh, perubahan nilai dari 0 ke 1 pada SFR dapat mengubah tegangan pada salah satu kaki MCU dari 0 V ke 3V. Beberapa *register* dapat ditulis saja (*write-only*), dibaca saja (*read-only*), atau dapat ditulis dan dibaca (*read/write*). Buku ini menggunakan MCU 32-bit yang artinya ukuran *register-register* yang dimiliki adalah 32-bit.

1.3. ARM

Acorn RISC *Machine/Advanced* RISC *Machine* (ARM) merupakan arsitektur prosesor 32/64-bit yang saat ini semakin banyak digunakan di industri. Dengan harga yang relatif sama dengan mikrokontroler 8-bit, prosesor berbasis ARM menawarkan kinerja yang jauh lebih tinggi dan kemampuan yang lebih lengkap. ARM merupakan salah satu jenis arsitektur *Reduced Instruction Set Computer* (RISC) untuk prosesor yang dikembangkan oleh ARM Holdings, sebuah perusahaan yang berlokasi di Inggris. Arsitektur ARM banyak digunakan oleh beberapa perusahaan, antara lain Atmel, Analog Devices, Dallas Semiconductor, Freescale, Infineon, Intel, Fujitsu, Micro*chip*, National Semiconductor, NXP, NEC, Renesas, Cypress, Samsung, Texas Instruments, ST Microelectronics, Zilog, dan Nuvoton.

Dengan jutaan buah mikrokontroler berbasis ARM yang telah terjual, ARM saat ini menjadi arsitektur mikrokontroler yang dominan. Lebih dari 95% *wearable devices* menggunakan ARM, demikian juga ARM memimpin dalam industri otomotif. Tiga kunci dari pengembangan ARM adalah ukuran kode yang kecil, kemampuan yang tinggi, dan konsumsi daya yang rendah. Dengan harga yang relatif sama, ARM 32-bit memberikan kinerja yang jauh lebih tinggi dibandingkan mikrokontroler 8-bit.

ARM merupakan salah satu arsitektur RISC dikarenakan memiliki fitur-fitur yang biasanya ada pada RISC, antara lain:

- Memiliki sebuah *Register File* yang besar dan seragam (*Register File* merupakan *array* dari *register* pada prosesor.)
- Merupakan arsitektur *load/store*, yakni operasi pemrosesan data hanya dilakukan pada isi dari *register*, bukan langsung dari isi memori
- Metode pengalamatan yang sederhana
- Instruksi memiliki panjang yang tetap (*fixed*) dan seragam

Sebagai tambahan, arsitektur ARM menyediakan juga:

- Kontrol atas *Arithmetic Logic Unit* (ALU) dan *shifter* pada sebagian besar instruksi pemrosesan data untuk memaksimalkan penggunaan ALU dan *shifter*
- Mode pengalamatan *auto-increment* dan *auto-decrement* untuk mengoptimalkan *program loop*
- Memuat (*load*) dan menyimpa (*store*) multi-instruksi untuk memaksimalkan *throughput* data
- Eksekusi secara kondisional (bersyarat) pada hampir semua instruksi untuk memaksimalkan *throughput* dari eksekusi program

Penyempurnaan dari RISC ini memungkinkan prosesor ARM mencapai keseimbangan di antara faktor-faktor kunci, yaitu performa yang tinggi, ukuran kode yang kecil, konsumsi daya yang rendah, dan ukuran *chip* yang kecil. Perbedaan ARM dengan beberapa arsitektur prosesor yang lain dapat dilihat pada Tabel 3.

Tabel 3. Perbandingan arsitektur 8051, PIC, AVR, dan ARM

	8051	**PIC**	**AVR**	**ARM**
Bus	8-bit	8/16/32-bit	8/32-bit	32-bit / 64-bit
Kecepatan	12 *Clock*/instruction cycle	4 *Clock*/instruction cycle	1 *Clock*/ instruction cycle	1 *Clock*/ instruction cycle
Arsitektur	*Von Neumann architecture*	*Harvard architecture*	*Modified*	*Modified Harvard architecture*
Konsumsi Daya	*Average*	*Low*	*Low*	*Low*
Contoh tipe	variasi 8051	PIC16,PIC17, PIC18, PIC24, PIC32	Tiny, Atmega, Xmega, *special purpose* AVR	ARMv7,8,9,10,11, Cortex
Harga relatif	*Very Low*	*Average*	*Average*	*Low*
MCU Populer	AT89C51, P89v51	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, beberapa jenis Arduino	ARM Cortex-M0 to ARM Cortex-M7

Ada banyak variasi dari ARM ini, misalnya ARM7, ARM8, ARM9, ARM10, ARM11, Cortex-A, Cortex-R, Cortex-M, dan sebagainya. Dibandingkan Cortex-A dan Cortex-R, ARM Cortex-M diperuntukkan

untuk aplikasi yang lebih sederhana dengan daya yang lebih rendah. Cortex-M0, Cortex M0+, dan Cortex-M23 diperuntukkan untuk aplikasi dengan biaya yang murah, daya rendah, dan luas area yang kecil. Cortex-M3, Cortex-M4, dan Cortex-M33 diperuntukkan untuk aplikasi yang menekankan keseimbangan antara performansi dan efisiensi energi. Cortex-M7 diperuntukkan untuk aplikasi yang memerlukan performansi tinggi.

Jenis ARM yang digunakan pada buku ini adalah Cortex-M4. ARM Cortex-M4 menawarkan beberapa fitur, di antaranya jumlah gerbang yang sedikit dengan proses yang cepat pada pemanggilan *interrupt* (*low latency*), menyediakan opsi *Floating Point Unit* (FPU), *Nested Vectored Interrupt Controller* (NVIC), *Memory Protection Unit* (MPU) dan beberapa fitur lainnya. Prosesor Cortex-M4 menggunakan antarmuka *Advanced High-performance Bus* (AHB)-Lite bus dan *Advanced Peripheral Bus* (APB).

Pada buku ini, MCU yang digunakan adalah STM32F407VGT6 yang merupakan *family* STM32F4. Tabel 4 menjelaskan arti *part number* tersebut.

Tabel 4. Penjelasan *part number* STM32F04VGT6

Kode	Arti
STM	Produsen: ST Microelectronics
32	MCU ARM 32-bit
F	*General purpose*
407	STM32F40xxx, *connectivity, camera Interface,* Ethernet
V	*Chip* 100 pin
G	1024 Kbytes of *flash memory*
T	*Chip package* LQFP
6	*Industrial temperature range*: -40 *to* 85 °C

1.4. STM32F407VGT6

STM32F407VGT6 merupakan sebuah mikrokontroler berbasis 32-bit ARM® Cortex®-M4 dengan *Flash* memori 1-Mbyte dan RAM 192-Kbyte dalam kemasan LQFP100 yang dibuat oleh ST Microelectronics, sebuah perusahaan berbasis Jenewa, Swiss. Mikrokontroler ini merupakan family dari STM32, yang berbasiskan ARM Cortex-M 32-

bit. Dengan kecepatan *clock* CPU mencapai 168 MHz, STM32F407VGT6 ini ditujukan untuk aplikasi yang memerlukan high performance. Beberapa fitur yang dimiliki antara lain:

- Frekuensi mencapai 168 MHz
- Flash memory: 1 Mbyte
- SRAM: 192+4 Kbytes
- Tegangan suplai dan I/O: 1.8 V s.d 3.6 V
- *Crystal* eksternal: 4 s.d 26 MHz
- Osilator RC internal: 16 MHz, dengan akurasi 1%
- Osilator internal 32 kHz untuk RTC
- Osilator RC internal 32 kHz
- 3 unit *Analog to Digital Converter* (ADC) 12-bit dengan kecepatan 2,4 MSPS
- 2 unit Digital to Analog Converter (DAC) 12-bit
- 17 buah *Timer*
- 140 buah I/O dengan kemampuan *interrupt*
- 3 unit Inter Integrated Circuit (I2C)
- 4 buah USART /2 UART
- 3 unit Serial Peripheral Interface (SPI)

Berikut penjelasan *memory*, *booting*, *clock*, pin I/O, dan fungsi alternatif pin yang dimiliki oleh STM32F407VGT6:

Memory

Pada STM32F4, *program memory, data memory, register* dan *port I/O* disusun dalam alamat-alamat dengan ukuran 4 GB, yang dibagi dalam empat blok berukuran 512 MB. Memori untuk kode program dimulai pada alamat 0x0000 0000, sementara memori untuk data (SRAM) dimulai pada alamat 0x2000 0000. Detail pengalamatan dapat dilihat pada "Table 1. STM32F4xx *register* boundary addresses" yang terdapat dalam *Reference Manual*.

Booting

STM32F4xx memiliki tiga mode *boot,* yakni main *flash memory*, *system memory*, dan *embedded* SRAM. Mode ini dapat dipilih melalui *BOOT*[1:0] sesuai dengan Tabel 5.

Tabel 5. Mode *booting* pada STM32F4xx

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

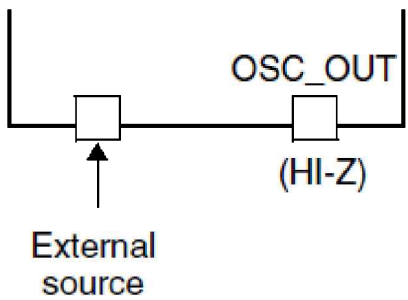
Distribusi *Clock*

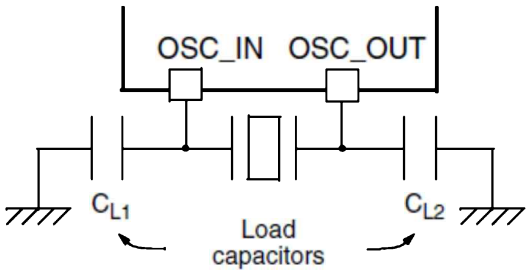
Gambaran *clock* pada STM32F407VGT6 dapat dilihat pada Gambar 3. Namun secara lengkap, diagram *clock* dapat juga dilihat pada STM32F4xx *Reference Manual*. *System Clock* (SYSCLK) dapat diatur oleh 5 sumber berbeda, yakni:

1) *High Speed Internal* (HSI) *oscillator clock*

HSI dihasilkan oleh RC osilator internal 16 MHz dan dapat langsung digunakan sebagai *System clock* maupun sebagai *input* PLL. Dengan HIS internal ini, maka komponen eksternal tidak diperlukan, sehingga dapat menghemat biaya.

2) *High Speed External* (HSE) *oscillator clock* (Gambar 2)



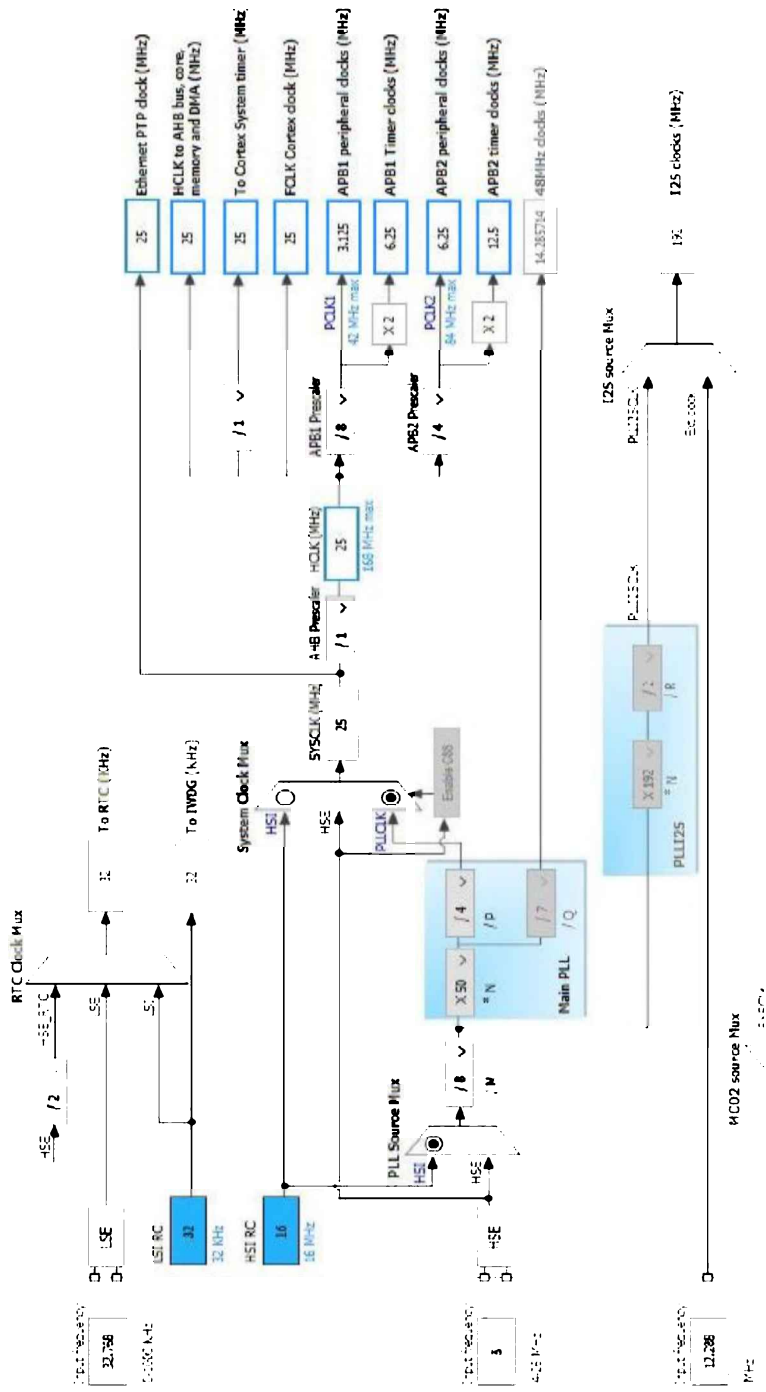


Gambar 2. *Clock* eksternal

3) *Main Phase Locked Loop* (PLL) *clock*

4) *Low Speed Internal* (LSI) RC internal dengan frekuensi 32 kHz

5) *Low Speed External* (LSE) dengan frekuensi 32,768 kHz untuk mengendalikan *Real Time Clock* (RTC)

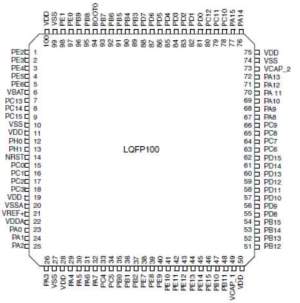


Gambar 3. Diagram *clock* pada STM32F407VGT6

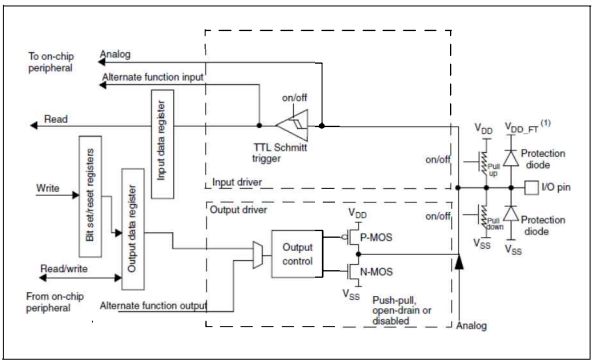
Pin *Input output* (I/O)

Buku ini menggunakan STM32F407VGT dengan package LQFP100 dengan total pin sebanyak 100 buah (Gambar 4). Pin-pin pada STM32F4xx dapat dikonfigurasi menjadi beberapa fungsi (Gambar 5), yakni:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain with pull-up or pull-down capability
- Output push-pull with pull-up or pull-down capability
- Alternate function push-pull with pull-up or pull-down capability
- Alternate function open-drain with pull-up or pull-down capability



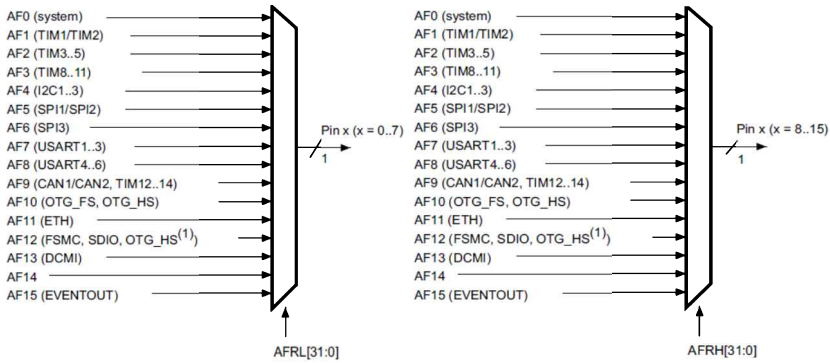
Gambar 4. Pin pada LQFP100



Gambar 5. Diagram I/O pin

Fungsi alternatif pin

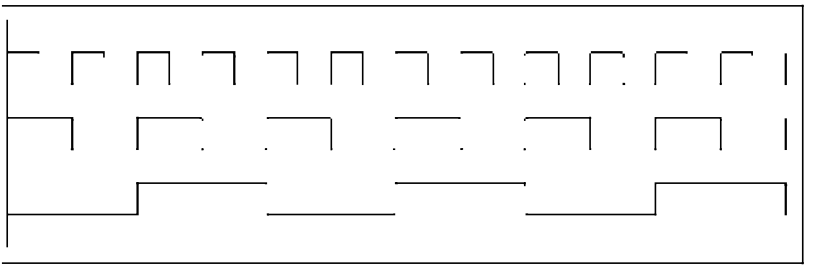
Pin-pin I/O dapat difungsikan untuk hal lain dengan cara menggunakan *multiplexer*. Fungsi alternatif pin-pin tersebut dapat dilihat pada Gambar 6. Pengguna dapat memilih dengan cara mengatur *register* AFRL an AFRH.



Gambar 6. *Multiplexer* untuk mengatur fungsi pin-pin

1.5. Latihan Soal Bab 1

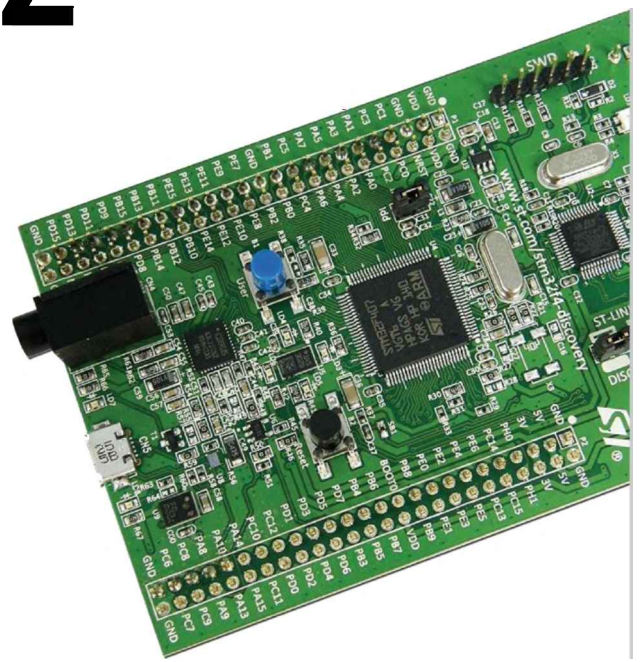
- Apa yang dimaksud dengan mikrokontroler?
- Sebutkan perbedaan antara mikrokontroler dan mikroprosesor!
- Anda ingin membuat sebuah sistem *embedded*. Sebutkan 3 hal yang menjadi pertimbangan dalam memilih sebuah mikrokontroler untuk sistem e*mbedded* tersebut! (Jelaskan secara spesifik)
- Sebutkan keunggulan arsitektur ARM dibandingkan dengan arsitektur lainnya!
- Sebutkan sumber *clock* yang dimiliki STM32F407VGT6!
- Apa yang dimaksud dengan fungsi alternatif pin?
- Apa saja yang harus dimiliki oleh sebuah sistem *embedded*?
- Sebutkan komponen dalam sebuah mikrokontroler!
- Sebutkan sumber *clock* pada MCU!
- Apa saja kelebihan MCU 32-bit berbasis ARM dibandingkan MCU 8 bit (misalnya AVR)?
- Perhatikan 3 (tiga) buah *clock* di bawah ini. Jika frekuensi *clock* yang paling atas 100 MHz, berapa MHz frekuensi *clock* yang paling bawah?



- Data pada ROM akan hilang ketika power dimatikan. Apakah pernyataan tersebut benar?
- Data pada RAM akan hilang ketika power dimatikan. Apakah pernyataan tersebut benar?

Sistem Embeded Berbasis ARM Cortex-M

2



Bab 2: Development Environment

Bab ini membahas mengenai *environment* yang digunakan pada buku ini, yakni *Embedded* C, *Hardware Abstraction Layer* (HAL), Cortex *MicroController Software Interface Standard* (CMSIS), STM32F4DISCOVERY, dan *Integrated Development Environment* (IDE) berupa Keil µVision dan STM32CubeMX.

2.1. Embedded C

Embedded C merupakan perluasan dari bahasa pemrograman C. Perluasan ini dibuat oleh C *Standards Committee* untuk mengatasi masalah kompatibilitas C untuk MCU yang berbeda-beda. Secara sintaks, pada dasarnya *Embedded* C sama dengan bahasa pemrograman C biasa. Fitur-fitur tambahan yang dimiliki *Embedded* C memudahkan pengguna dalam mengoperasikan beberapa hal, seperti *fixed-point arithmetic*, pengaturan memori, dan operasi I/O. Tipe-tipe bilangan yang digunakan dapat dilihat pada Tabel 6.

Tabel 6. Tipe bilangan

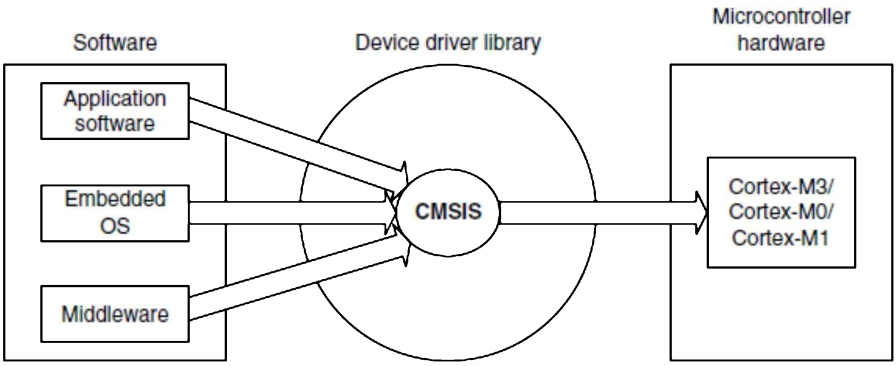
C type	stdint.h type	Bit	Sign	Range
char	uint8_t	8	Unsigned	0 s 255
signed char	int8_t	8	Signed	-128 s.d 127
unsigned short	uint16_t	16	Unsigned	0 s.d 65535
short	int16_t	16	Signed	-32768 s.d 32767
unsigned int	uint32_t	32	Unsigned	0 s.d 4294967295
int	int32_t	32	Signed	-2147483648 s.d 2147483647
unsigned long long	uint64_t	64	Unsigned	0 s.d 18446744073709551615
long long	int64_t	64	Signed	-9223372036854775808 s.d 9223372036854775807

C type	stdint.h type	Bit	Sign	Range
float	float	32	Signed	-3,4E38 s.d 3,4E38
double	float	64	Signed	-1,7E308 s.d 1,7E308

2.2. HAL dan CMSIS

Hardware Abstraction Layer (HAL) merupakan suatu *layer* dari pemrograman yang memungkinkan sebuah *Operating System* (OS) berinteraksi dengan *hardware* dengan menggunakan bahasa level tinggi atau abstrak dibandingkan dengan menggunakan bahasa level rendah yang detail. Dengan demikian, *programmer* dapat menulis program tanpa harus mengerti detail *hardware*-nya. Keuntungan menggunakan HAL ini adalah kode yang dihasilkan *portable* dan *reusable*, menghemat biaya, dan meminimalisir *bug*. Pada dasarnya, HAL merupakan *Application Program Interface* (API) yang didesain untuk interaksi dengan *hardware*.

Cortex *MicroController Software Interface Standard* (CMSIS) merupakan HAL untuk prosesor Cortex-M yang bersifat *vendor-independent* (tidak terikat dengan *vendor*). CMSIS dikembangkan oleh ARM pada tahun 2008 untuk memungkinkan semua *resource* ini digunakan untuk memrogram mikrokontroler ARM dengan mudah dan cepat. Dengan adanya CMSIS, maka kini memungkinkan dukungan terhadap prosesor Cortex-M secara konsisten, *interface* yang sederhana ke prosesor dan periferal, menyederhanakan penggunaan kembali (*re-use*) *software*, mengurangi *learning curve* untuk pengembang MCU, dan mengurangi waktu untuk pemasaran peralatan baru – yang pada akhirnya akan menghemat biaya. Fungsi CMSIS dapat dilihat pada Gambar 7.



Gambar 7. Fungsi CMSIS (Gambar: Yiu, 2010)

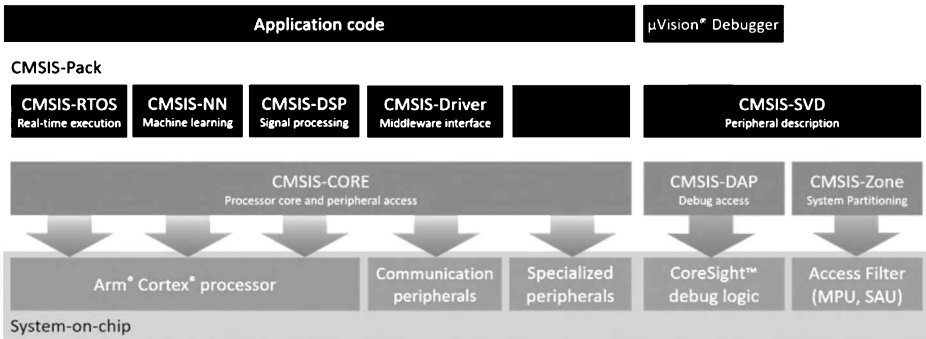
CMSIS memiliki standardisasi yang melingkupi:

- *Hardware Abstraction Layer* (HAL) untuk *register* pada Cortex-M processor *registers* melingkupi:
- Nested Vectored Interrupt Controller (NVIC)
- System Control Block registers
- SYSTICK register, MPU registers
- Sejumlah fungsi NVIC dan *core feature access*
- *System exception names* yang terstandardisasi: Ini memungkinkan OS dan *middleware* menggunakan *system exception* tanpa masalah kompatibilitas.
- Metode untuk pengorganisasian *header file* yang terstandardisasi
- Metode untuk inisialisasi sistem yang terstandardisasi
- Fungsi-fungsi intrinsik yang terstandardisasi
- Fungsi-fungsi akses untuk komunikasi
- Cara menentukan frekuensi *clock* yang terstandardisasi

Gambar 8 menjelaskan komponen-komponen yang disediakan CMSIS antara lain:

- CMSIS-*Core*: Cortex-A dan Cortex-M
- CMSIS-*Driver*
- CMSIS-DSP: *Digital* Signal Processing
- CMSIS-NN: Neural Network
- CMSIS-RTOS: Real Time Operating System
- CMSIS-Pack
- CMSIS-SVD: System View Description
- CMSIS-DAP: *Debug Access* Port

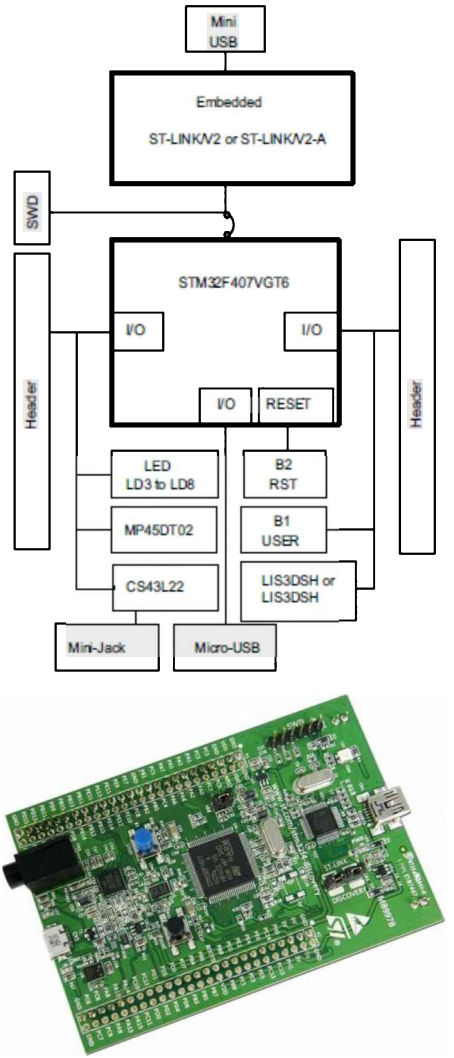
- CMSIS-Zone: Definition *and partitioning*



Gambar 8. Struktur CMSIS (Gambar: www.keil.com)

2.3. Kit STM32F407G-DISC1

STM32F407G-DISC1 (STM32F4DISCOVERY) merupakan salah satu *development kit* yang dikembangkan oleh STMicroelectronics dengan tujuan untuk memudahkan pengguna dalam membuat aplikasi berbasis mikrokontroler STM32F407. Kit ini sudah mengandung ST-LINK, alat yang digunakan untuk melakukan proses *debugging* dan mengunggah program ke dalam mikrokontroler. Selain itu, kit ini juga mengandung *accelerometer digital,* mikrofon *digital*, *Digital to Analog Converter* (DAC), LED, *push button*, dan konektor untuk USB *On-The-Go* (OTG). STMicroelectronics merilis versi terbaru dari kit ini dengan kode pemesanan STM32F407G-DISC1.



Gambar 9. STM32F4DISCOVERY

Fitur-fitur yang dimiliki kit ini adalah:

- Mikrokontroler STM32F407VGT6 (32-bit ARM® Cortex®-M4 dengan *Flash* memori 1-Mbyte dan RAM 192-Kbyte dalam kemasan LQFP100)
- ST-LINK / V2-A yang terintegrasi dalam kit
- USB ST-LINK dengan tiga *interface* yang berbeda, yakni *debug port*, *virtual COM port*, dan *mass storage*

- Pilihan suplai daya yang dapat menggunakan USB atau ekternal 5V
- ST MEMS 3-*axis accelerometer*: LIS302DL atau LIS3DSH
- Mikrofon *digital* MP45DT02 ST-MEMS
- DAC audio CS43L22 dengan pengeras suara kelas D yang terintegrasi
- Delapan buah LED:
    - LD1 (merah/hijau) untuk komunikasi USB
    - LD2 (merah) untuk indikasi bahwa 3.3 V ON
    - Empat LED untuk aplikasi pengguna, yakni LD3 (jingga), LD4 (hijau), LD5 (merah) *and* LD6 (biru)
    - Dua LED untuk USB OTG, yakni LD7 (hijau, untuk VBUS) dan LD8 (merah, untuk arus berlebih)
- Dua *push button*, yakni satu tombol untuk reset dan satu tombol untuk aplikasi pengguna
- Konektor untuk seluruh kaki STM32F407VGT6 (100 kaki)
- Dukungan perangkat lunas gratis seperti STM32CubeF4 atau STSW-STM32068, termasuk beberapa contoh program

Perangkat lunak yang dapat digunakan bersamaan dengan kit ini adalah IAR EWARM (IAR *Embedded* Workbench), Keil MDK-ARM, dan GCC-based IDE (misalnya Free AC6: SW4STM32, Atollic, dan TrueSTUDIO).

2.4. Integrated Development Environment (IDE)

IDE merupakan aplikasi *software* yang menyediakan fasilitas komprehensif kepada *programmer* untuk pengembangan *software*nya. Biasanya IDE terdiri atas:

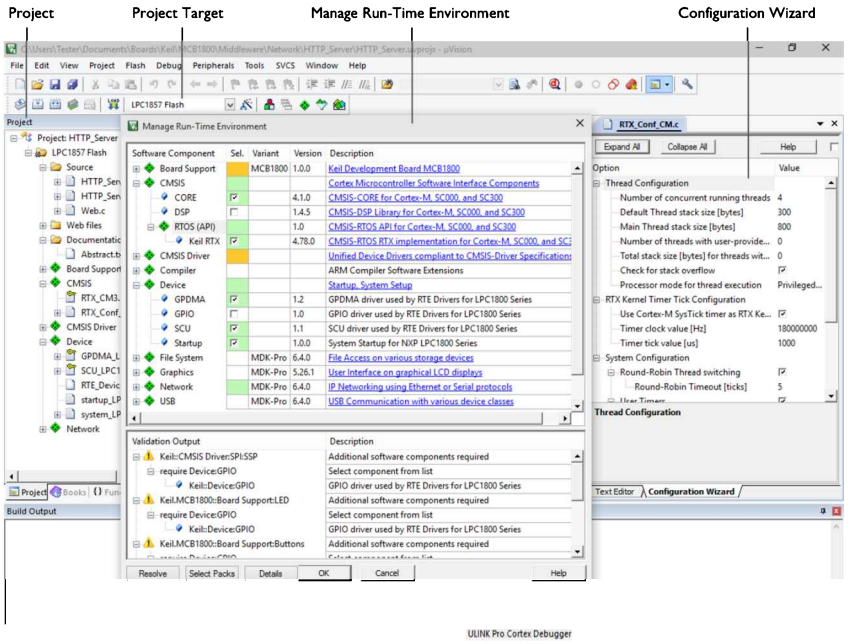
- Source code editor
- Compiler dan/atau intepreter
- Build tools
- Programmer
- Debugger

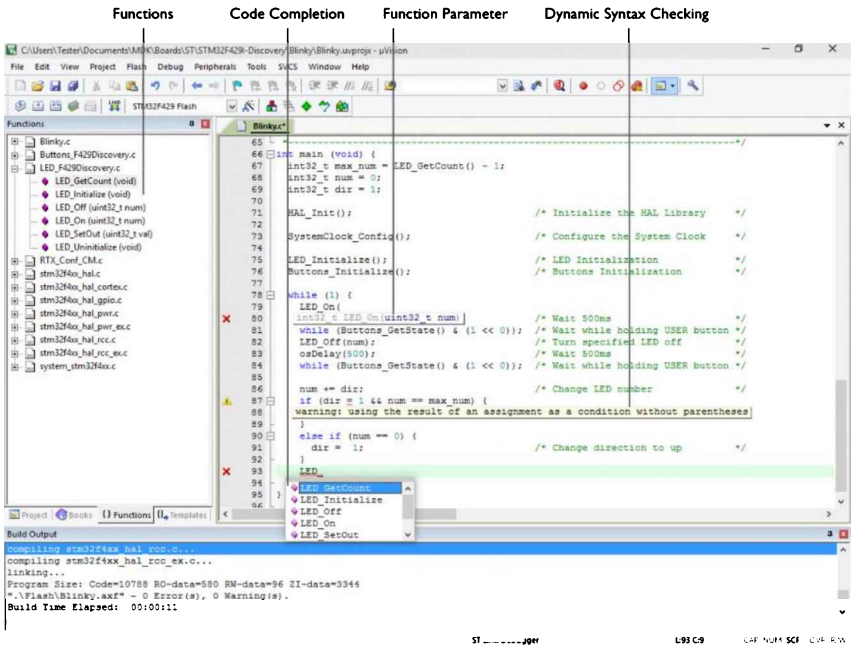
Walaupun demikian, tiap IDE memiliki fitur-fitur yang berbeda. IDE yang kita gunakan dalam buku ini adalah Keil µVision, dibantu dengan STM32CubeMX.

Keil µVision

Ada beberapa perangkat lunak yang dapat digunakan untuk menjalankan STM32F4. Pada buku ini, Keil µVision (ARM Germany GmbH, Grasbrunn, Jerman) akan digunakan untuk membuat proyek-proyek sederhana. Keil merupakan IDE yang sudah secara luas digunakan di seluruh dunia sejak tahun 1986. Dengan banyaknya *chip* yang didukung, sampai ini Keil masih menjadi andalan bagi banyak pemrogram mikrokontroler.

µVision mengombinasikan *project management*, *run-time environment*, *Build facilities, Source code editor* dan program *debugg*ing dalam satu *Software*. Dengan menggunakan *debugger*, *programmer* dapat menguji, memverifikasi, dan mengoptimasi kode pemrograman. *Debugger* yang dimiliki µVision memiliki fitur *breakpoints*, *watch windows* dan *execution control,* dan menyediakan pula akses untuk memonitor periferal. Selain itu, Keil juga menyediakan *link* ke manual penggunaan, *online help*, dan *datasheet* beberapa MCU.





Gambar 10. Bagian-bagian Keil µVision

Versi Keil µVision yang dipakai dalam buku ini adalah adalah V5.24.2.0, yang dapat diunduh secara gratis pada http://www.keil.com/. Versi gratis ini memiliki beberapa keterbatasan, di antaranya ukuran kode maksimal 32 Kbyte. Sistem Operasi yang dapat digunakan adalah Windows (Windows 7, 8, 8.1, 10). Persyaratan perangkat keras komputer minimal untuk menginstalasi Keil µVision v5 adalah sebagai berikut:

- Prosesor: 32-bit atau 64-bit 1 GHz
- RAM 1 GB
- Tersedia minimal 2 GB ruang di dalam *harddisk*
- Jaringan internet untuk *product update* melalui PackInstaller

STM32CubeMX

Melakukan konfigurasi awal (inisialisasi) pada mikrokontroler merupakan pekerjaan yang sulit, terutama bagi pemula. Untuk itu, STMicroelectronics menyediakan sebuah perangkat lunak yang dinamakan STM32CubeMX (STMicroelectronics, Jenewa, Swiss),

yang merupakan bagian dari STMCube™ yang dirancang untuk memungkinkan pemrogram melakukan pemrograman dengan lebih mudah dan menghemat waktu serta biaya. STM32CubeMX ini dibuat untuk memudahkan pemrogram dalam melakukan konfigurasi awal pada mikrokontroler karena konfigurasi tersebut dilakukan secara grafis. Dari konfigurasi grafis ini dihasilkan kode program dalam bahasa C secara otomatis. Buku ini menggunakan STM32CubeMX Version 4.23.0, yang dapat dijalankan pada sistem operasi Windows, Linux, MacOS, maupun Eclipse. Perangkat lunak ini dapat diunduh secara gratis; Kita hanya diminta konfirmasi email.

Fitur yang dimiliki oleh STM32CubeMX adalah sebagai berikut:

- Database mikrokontroler yang cukup banyak
- Pilihan untuk mengonfigurasi mikrokontroler secara grafis/visual beberapa parameter berikut:
    - Konfigurasi *Pinout*
    - Bagan/diagram *clock*
    - Pengaturan parameter pada periferal
    - Prediksi konsumsi daya

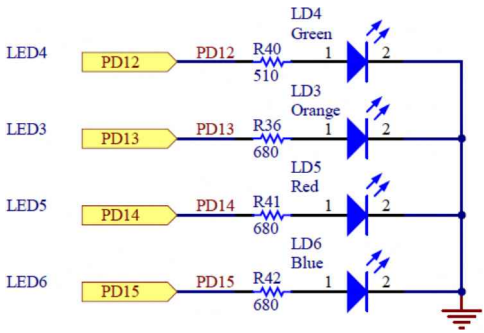
Berdasarkan konfigurasi yang telah dibuat, STM32CubeMX dapat secara otomatis menghasilkan kode C yang dapat dibuka dengan menggunakan *Compiler* IAR™, Keil™, ataupun GCC.

2.5. Praktikum Bab 2

Alat dan Bahan

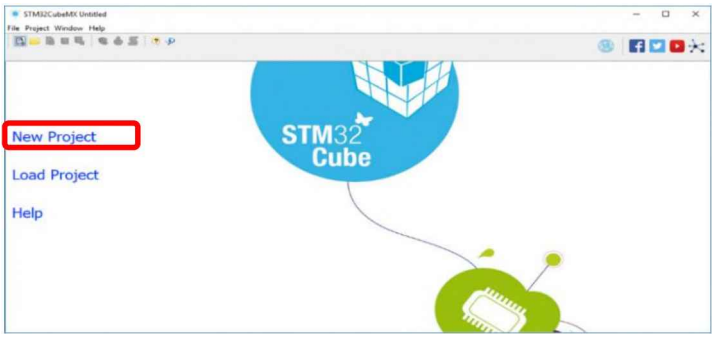
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Pada praktikum ini, kita akan menyalakan LED dengan program yang sederhana. STM32F4DISCOVERY memiliki beberapa lampu LED *on board* yang dapat kita gunakan. Sebagai contoh LED3, LED4, LED5, dan LED6 yang masing-masing terhubung pada PD13, PD12, PD14, dan PD15.

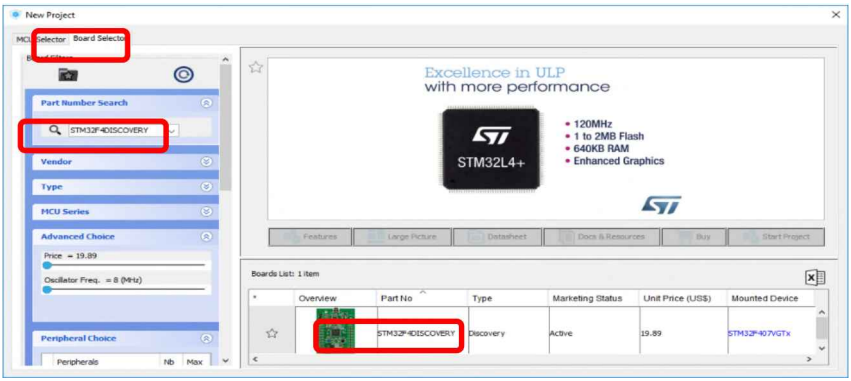


Kita akan menyalakan LED tersebut menggunakan STM32CubeMX dan Keil µVision. Jalankan langkah-langkah berikut:

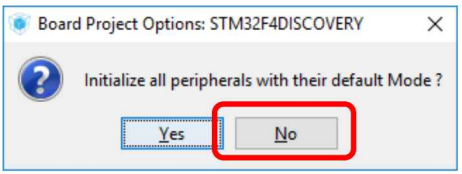
- Jalankan STM32CubeMX, kemudian klik New Project.



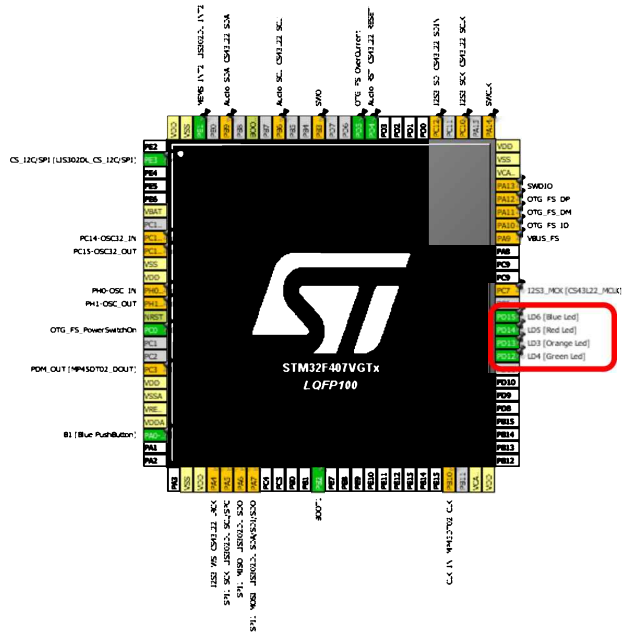
- Pilih *tab* Board Selector. Pada Part Number Search, pilih STM32F4DISCOVERY. Klik ganda STM32F4DISCOVERY pada Board List.



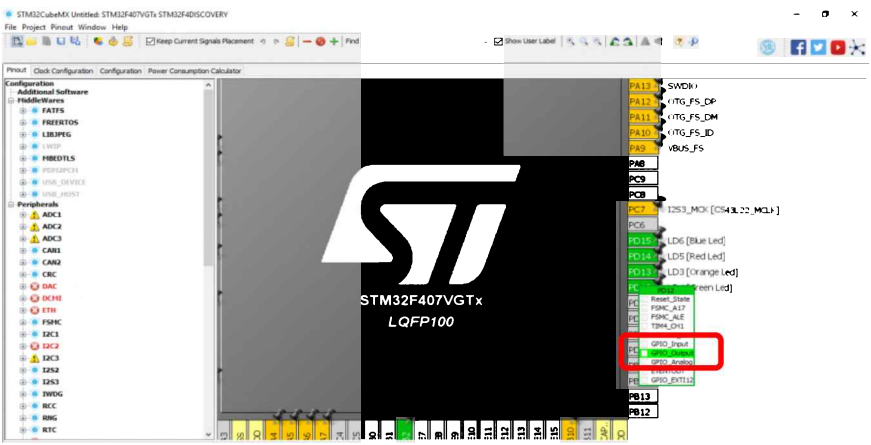
- Klik No saat muncul tampilan berikut:



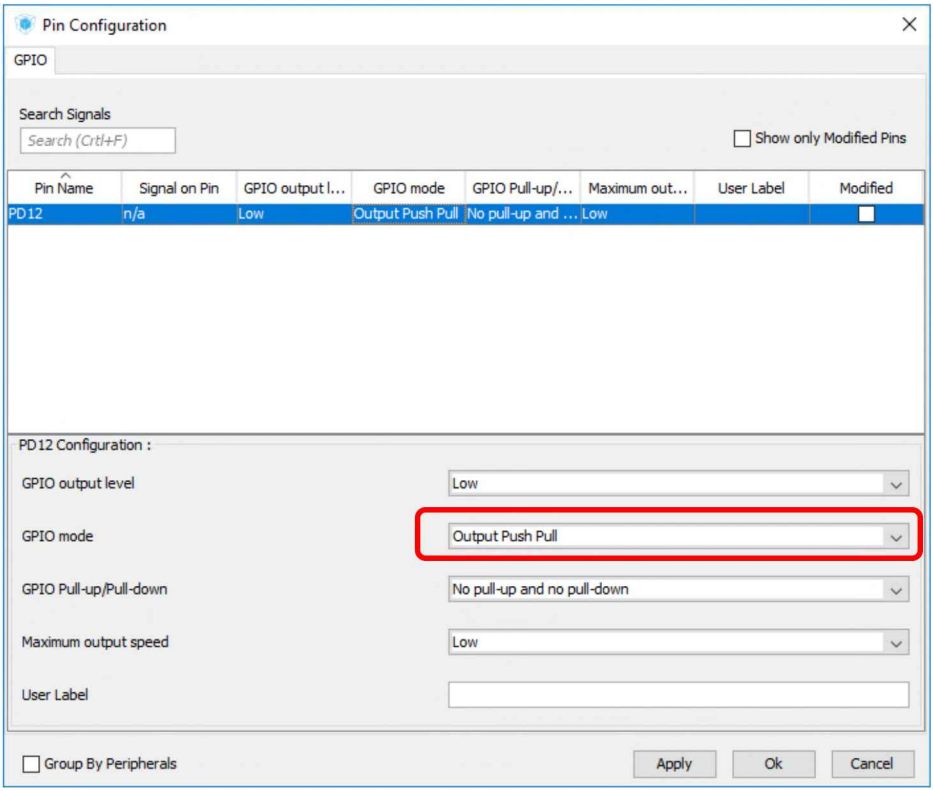
- Pemetaan pin akan muncul. Dari gambar ini akan terlihat penggunaan masing-masing pin pada mikrokontroler yang sudah terkait dengan board STM32F4DISCOVERY. Perhatikan LD3, LD4, LD5, dan LD6.



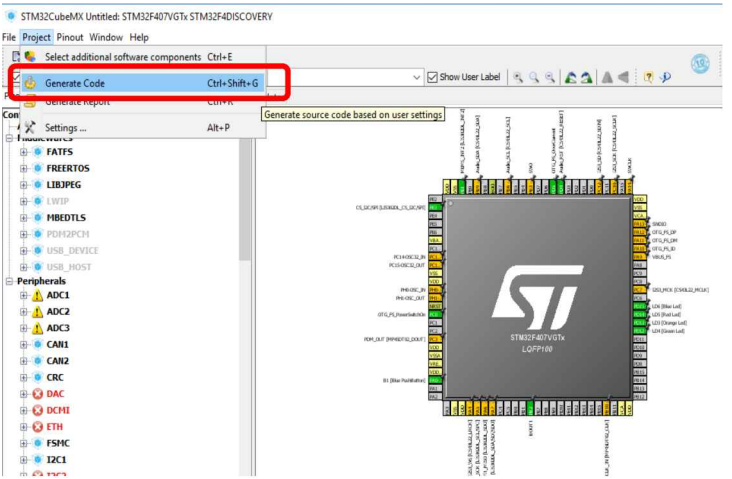
- Klik PD12 dan jadikan sebagai *output* (GPIO_Output). PD12 ini terhubung pada LD4 (LED hijau).



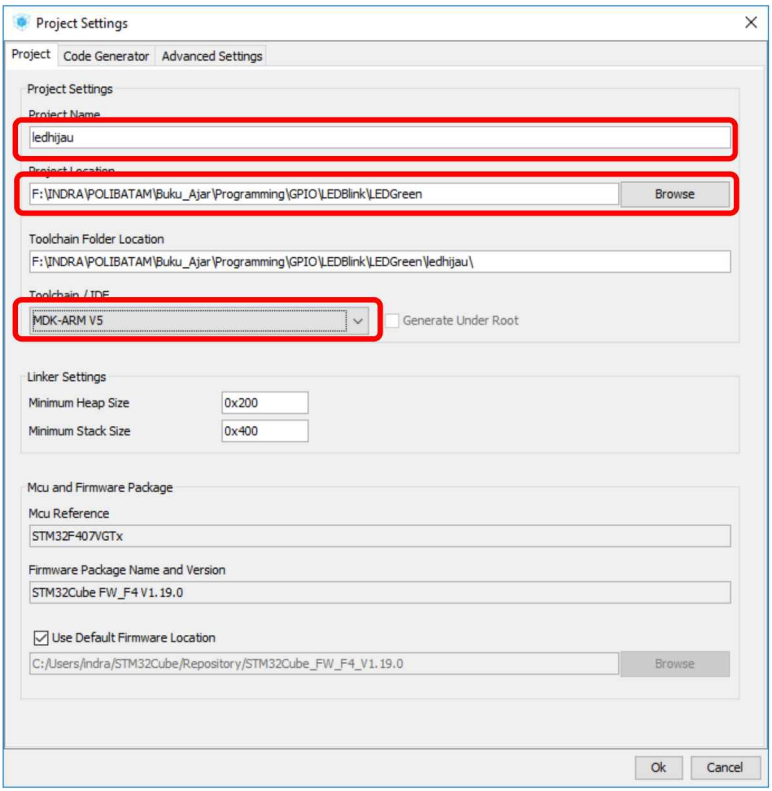
- Pada Pin Configuration, jadikan PD12 sebagi Output Push Pull. Klik OK.



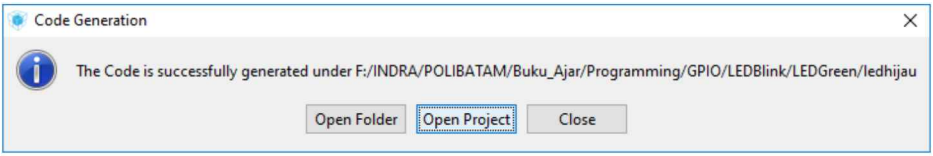
- Klik Project | Generate Code.

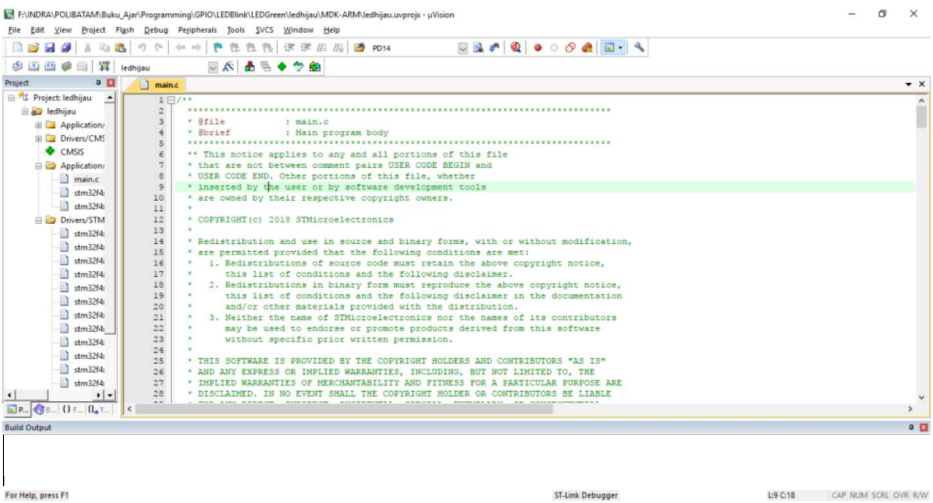


- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.



- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.





- Buka *file tab* main.c. Di dalamnya sudah terset secara otomatis dari *port* D. Dalam kode tersebut, LD4 (PD12) yang mengendalikan LED hijau sudah diset sebagai *output*.

```
/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

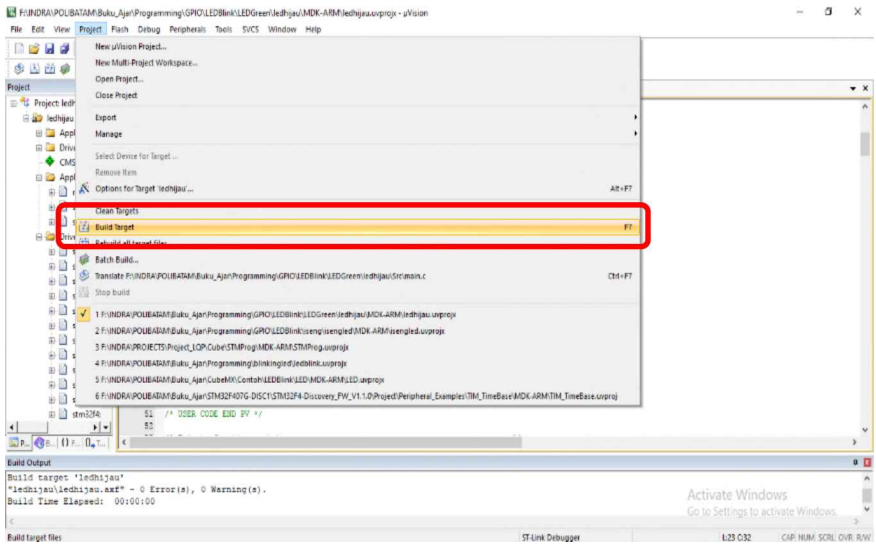
Juga tertulis:

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin, GPIO_PIN_RESET);
```

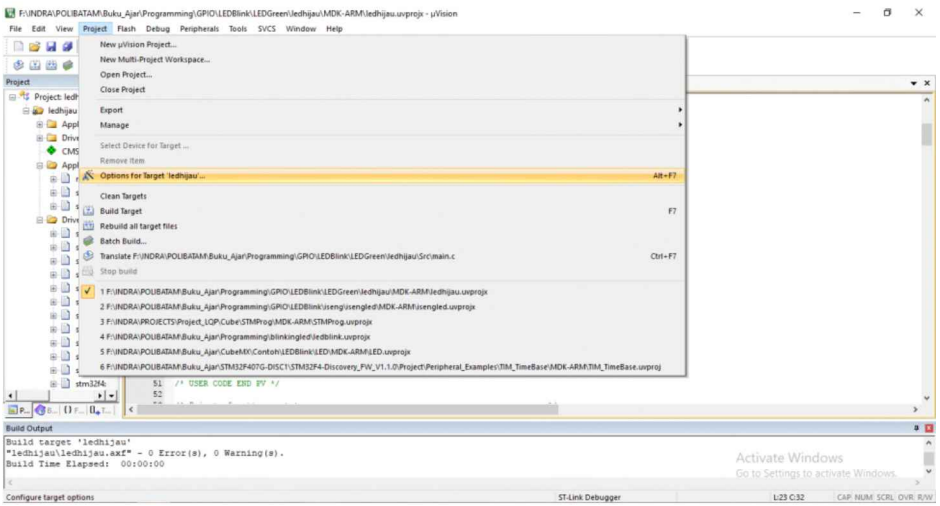
- Temukan fungsi int main(), kemudian masukkan empat baris kode di bawah pada blok kode while(1).

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  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();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        HAL_GPIO_WritePin(GPIOD, LD4_Pin, GPIO_PIN_RESET);
        HAL_Delay(1000);
        HAL_GPIO_WritePin(GPIOD, LD4_Pin, GPIO_PIN_SET);
        HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
```

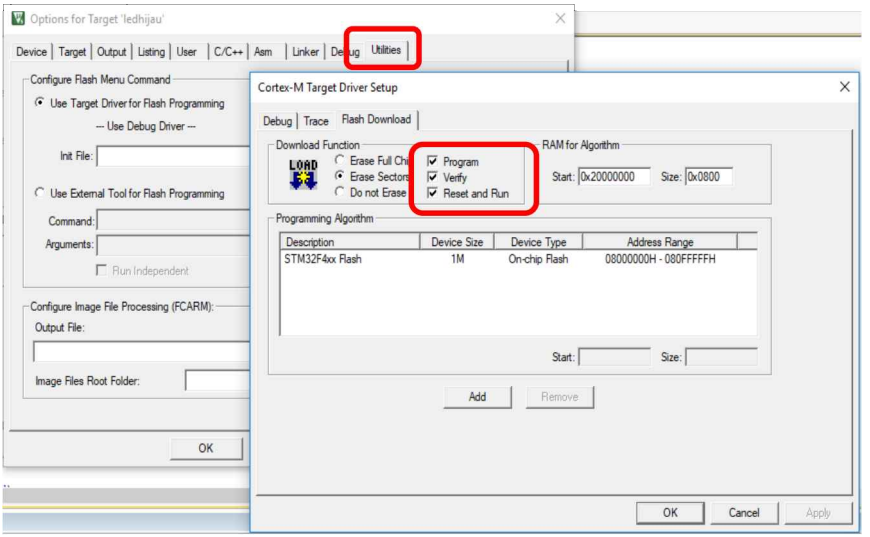
- *Compile* kode tersebut dengan cara klik Project | *Build* Target (atau dengan menekan icon  maupun menekan tombol F7 pada keyboard), kemudian tunggu hingga prosesnya selesai.



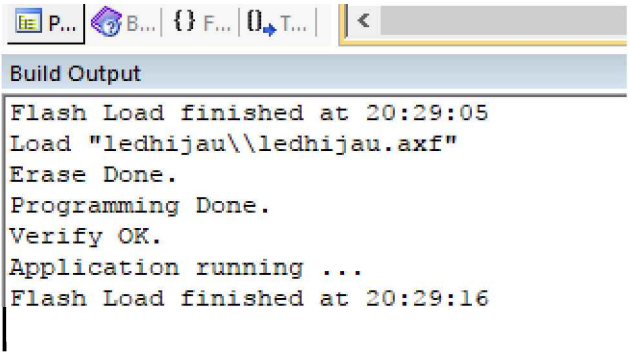
- Setelah proses compile selesai, set STLink *programmer* dengan cara klik Project | Options for Target 'Nama File' (atau dengan klik icon  maupun menekan tombol ALT+F7 pada keyboard).



- Pilih *tab* Utilities dan set parameter seperti gambar di bawah. Kemudian klik OK.



- Sekarang, load program yang telah dibuat ke dalam mikrokontroler. Pastikan STM32F4DISCOVERY sudah terhubung dengan computer via USB. Klik Flash | Download (atau dengan menekan icon  maupun menekan tombol F8 pada keyboard). Jika proses berhasil, maka akan muncul tulisan berikut:



- Perhatikan LED hijau STM32F4DISCOVERY. LED tersebut akan berkedip dengan frekuensi 1 Hz.

2.6. Tugas Praktik Bab 2

- Lakukan hal yang sama dengan praktikum, namun ganti LED hijau dengan LED biru!

- Lakukan hal yang sama dengan praktikum, namun ganti LED hijau dengan LED merah!

2.7. Latihan Soal Bab 2

- KEIL µVision yang biasa kita gunakan pada dasarnya merupakan…
- Apa yang dimaksud dengan IDE?
- Apa saja kelebihan-kelebihan yang dimiliki STM32F4DISCOVERY?
- Sebutkan keuntungan menggunakan STM32CubeMX dibandingkan memrogram mikrokontroler *from scratch*!
- Hitung nilai c sesuai operasi pada tabel berikut!

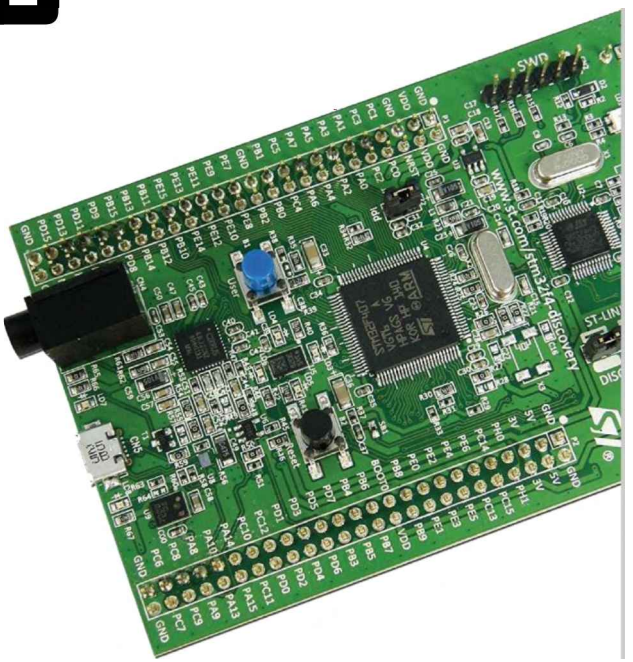
No	uint8_t a	uint8_t b	uint8_t c
1	13	5	c = a % b
2	13	5	c = a * b
3	13	5	c = a + b
4	13	5	c = a - b
5	13	5	c = a / b
6	217	170	c = a & b
7	217	3	c = a << b
8	217	3	c = a >> b
9	217	170	c = a ^ b
10	217	170	c = a | b
11	217		c = ~ a

- Hitung nilai a sesuai operasi pada tabel berikut!

No	uint8_t a	
1	170	a &= 1
2	170	a *= 2
3	170	a++
4	170	++a
5	170	a += 2
6	170	a --
7	170	--a
8	170	a -= 2
9	170	a /= 3
10	170	a |= 1

Sistem Embeded Berbasis ARM Cortex-M

3



Bab 3 GPIO

Mikrokontroler memiliki *General Purpose Input output* (GPIO) yang berfungsi untuk mengontrol nilai tegangan di beberapa kakinya atau membaca nilai tegangannya, baik HIGH maupun LOW. Perhatikan Tabel 7.

Tabel 7. Level logika

Logic level	Active-high signal (Q)	Active-low signal ($\bar{Q}$)
Logical HIGH	1	0
Logical LOW	0	1

3.1. Konfigurasi GPIO

Setiap *port* GPIO pada STM32F4 memiliki empat buah *register* 32-bit untuk konfigurasi (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR dan GPIOx_PUPDR), dua buah *register* data 32-bit (GPIOx_IDR dan GPIOx_ODR), sebuah *register set/reset* 32-bit (GPIOx_BSRR), sebuah *register locking* 32-bit (GPIOx_LCKR), dan buah *register* untuk memilih fungsi alternatif 32-bit (GPIOx_AFRH dan GPIOx_AFRL). Perhatikan Tabel 8.

Tabel 8. Tabel konfigurasi GPIO

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]		I/O configuration	
01	0	SPEED [B:A]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1		0	0	GP output	OD
	1		0	1	GP output	OD + PU
	1		1	0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	

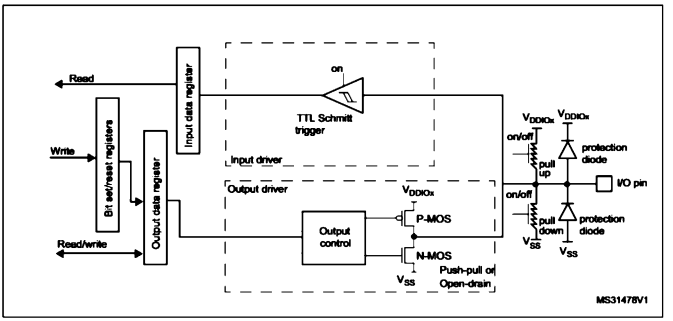
10	0	SPEED [B:A]		0	0	AF	PP
	0			0	1	AF	PP + PU
	0			1	0	AF	PP + PD
	0			1	1	Reserved	
	1			0	0	AF	OD
	1			0	1	AF	OD + PU
	1			1	0	AF	OD + PD
	1			1	1	Reserved	
00	x	x	x	0	0	Input	Floating
	x	x	x	0	1	Input	PU
	x	x	x	1	0	Input	PD
	x	x	x	1	1	Reserved (input floating)	
11	x	x	x	0	0	Input/output	Analog
	x	x	x	0	1	Reserved	
	x	x	x	1	0		
	x	x	x	1	1		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 11. GPIOx_MODER

Register GPIOx_MODER (x = A..I/J/K) digunakan untuk memilih konfigurasi I/O (*input*, *output*, AF, analog). Kita dapat melakukan konfigurasi sesuai Gambar 11. Diagram pin I/O pada STM32F4 dapat dilihat pada Gambar 12.



Gambar 12. Diagram pin I/O

3.2. Digital Output

Untuk menjadikan suatu I/O sebagai *output*, kita perlu memberikan nilai 01 pada MODER[1:0]. Setelah itu, kita dapat memilih tipe *output* yang kita inginkan dengan cara memberi nilai pada GPIO *port output type register* (GPIOx_OTYPER). Nilai 0 berarti *output push-pull (reset state)*, sedangkan nilai 1 berarti *output open-drain*.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 13. GPIOx_OTYPER

Selain itu, STM32F4xx menyediakan fitur untuk mengatur kecepatan dari tiap pin *output* tersebut, yakni dengan cara mengonfigurasi GPIO *port output speed register* (GPIOx_OSPEEDR) dengan nilai sebagai berikut:

- 00: Low speed
- 01: Medium speed
- 10: High speed
- 11: Very high speed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 14. GPIOx_OSPEEDR

Nilai *port* yang akan kita keluarkan diset melalui GPIO *port output data register* (GPIOx_ODR):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 15. GPIOx_ODR

3.3. Digital Input

Untuk menjadikan pin sebagai *input*, kita perlu memberikan nilai pada GPIOx_MODER menjadi 00. Nilai bacaan *input* kemudian disimpan di GPIO *port input data register* (GPIOx_IDR).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Gambar 16. GPIOx_IDR

STM32Fxx menyediakan fungsi *pull-up* dan *pull-down* internal. Hal ini dapat dilakukan dengan memberikan nilai pada GPIO *port pull-up/pull-down register* (GPIOx_PUPDR) pada Gambar 17. Nilai 00 berarti *pull-up,* sedangkan *pull-down* dinonaktifkan, 01 berarti *pull-up* aktif, dan 10 berarti *pull-down* yang aktif.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

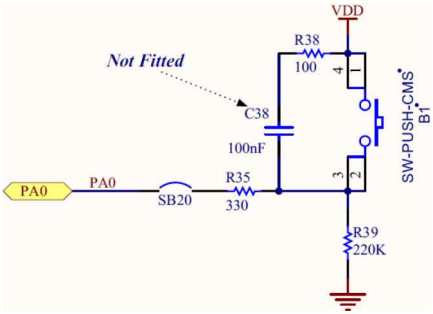
Gambar 17. GPIOx_PUPDR

3.4. Praktikum Bab 3

Alat dan Bahan

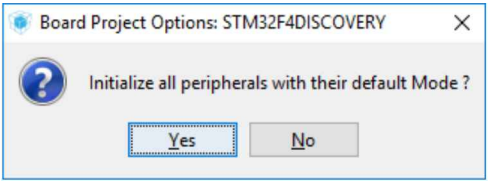
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan Driver STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Pada praktikum ini, kita akan mendemonstrasikan penggunaan *push button* pada STM32F4DISCOVERY. *Push button* yang kita gunakan adalah B1. Koneksi B1 ini dapat kita lihat pada skematik di gambar di bawah. Ketika *button* ditekan, maka PA0 akan memiliki logika 1, sedangkan jika dilepas akan berlogika 0 (*pull down* menggunakan resistor R39 sehingga *pull down* internal tidak diperlukan lagi).

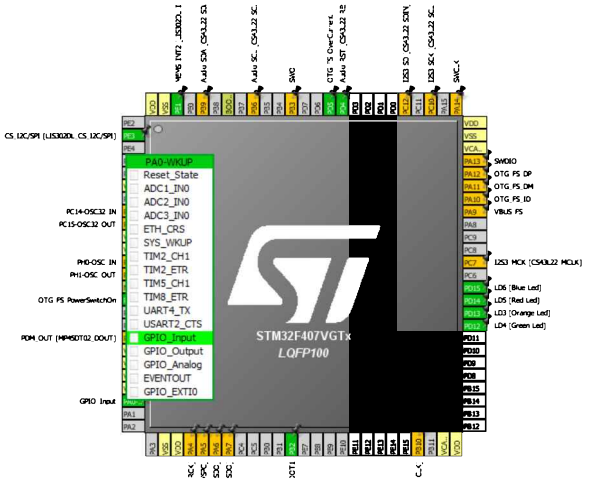


Tujuan program adalah menyalakan hanya LED biru ketika B1 ditekan dan menyalakan hanya LED hijau ketika B1 dilepas. Lakukan langkah-langkah berikut:

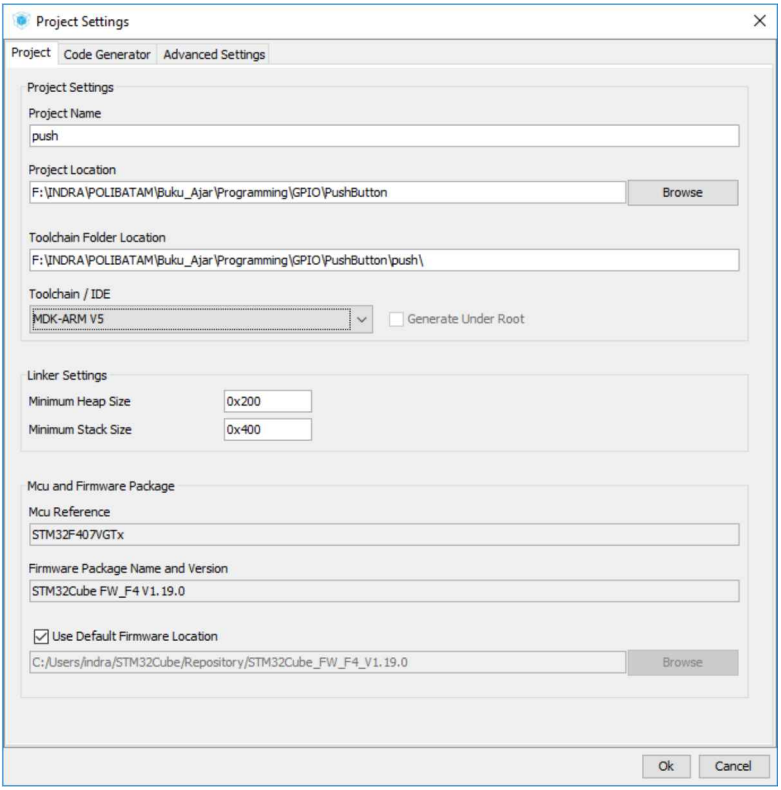
- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* Board Selector. Pada Part Number Search, pilih STM32F4DISCOVERY. Klik ganda STM32F4DISCOVERY pada Board List.
- Klik No saat muncul tampilan berikut:



- Pemetaan pin akan muncul. Dari gambar ini akan terlihat penggunaan masing-masing pin pada mikrokontroler yang sudah terkait dengan *board* STM32F4DISCOVERY. Set LD4 dan LD6 sebagai *output*. Klik PA0 dan set PA0 sebagai GPIO_Input. Lihat gambar di bawah.



- Klik Project | Generate Code.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.



- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c. Di dalamnya sudah terset secara otomatis *port* A dan *port* D. Dalam kode tersebut, LD4 (PD12) dan LD6 (PD15) sudah diset sebagai *output*, sedangkan PA0 sudah diset sebagai *input*.

```
/*Configure GPIO pin : PA0 */
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
                        Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

Juga tertulis:

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                         |Audio_RST_Pin, GPIO_PIN_RESET);
```

- Temukan fungsi int main(), kemudian masukkan empat baris kode di bawah pada blok kode while(1).

```
uint32_t NilaiPin;
while (1)
{
 NilaiPin = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
 if (NilaiPin == 0) // Tombol dilepas
 {
  HAL_GPIO_WritePin(GPIOD, LD4_Pin, GPIO_PIN_SET); // LED hijau ON
  HAL_GPIO_WritePin(GPIOD, LD6_Pin, GPIO_PIN_RESET); // LED biru OFF
 }
 else // Tombol ditekan
 {
  HAL_GPIO_WritePin(GPIOD, LD4_Pin, GPIO_PIN_RESET); // LED hijau OFF
  HAL_GPIO_WritePin(GPIOD, LD6_Pin, GPIO_PIN_SET); // LED biru ON
 }
}
```

- Compile kode tersebut dengan cara klik Project | Build Target (atau dengan menekan icon  maupun menekan tombol F7 pada keyboard), kemudian tunggu hingga prosesnya selesai.

- Setelah proses compile selesai, set STLink *programmer* dengan cara klik Project | Options for Target 'Nama *File*' (atau dengan klik icon  maupun menekan tombol ALT+F7 pada keyboard).
- Sekarang, load program yang telah dibuat ke dalam mikrokontroler. Pastikan STM32F4DISCOVERY sudah terhubung dengan computer via USB. Klik Flash | Download (atau dengan menekan icon 

maupun menekan tombol F8 pada keyboard).

Jika proses load program sukses, perhatikan LED hijau dan biru. Ketika B1 dilepas, maka hanya LED hijau yang menyala. Ketika B1 ditekan, maka hanya LED biru yang menyala. Pahami setiap baris kode program tersebut.

3.5. Tugas Praktik Bab 3

Tugas Praktik 3-01

Nyalakan LED3, LED4, LED5, dan LED6 dengan berurutan setiap detik sesuai dengan tabel di bawah:

Detik ke	LED Hijau	LED Jingga	LED Merah	LED Biru
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON
5	ON	OFF	OFF	OFF
Dan seterusnya	...	...	...	...

Tugas Praktik 3-02

Gunakan *push button* B1. Jika B1 dilepas, LED menyala dengan urutan seperti Tugas no.1 di atas. Namun, jika B1 ditekan, LED menyala berurutan setiap detik dengan arah sebaliknya, yakni seperti tabel di bawah:

Detik ke	LED Hijau	LED Jingga	LED Merah	LED Biru
1	OFF	OFF	OFF	ON
2	OFF	OFF	ON	OFF

3	OFF	ON	OFF	OFF
4	ON	OFF	OFF	OFF
5	OFF	OFF	OFF	ON
Dan seterusnya	...	...	...	...

3.6. Latihan Soal Bab 3

- Berapa nilai GPIOx_MODER agar

  PD15 = *OUTPUT*

  PD14 = *INPUT*

  PD13 = *OUTPUT*

  PD12 = *INPUT*

  PD[11:0] = *INPUT* ?

- Berapa nilai GPIOD_ODR agar kaki

  PD15 = 3,3V

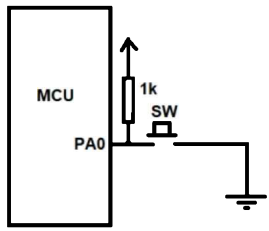
  PD14 = 0V

  PD13 = 3,3V

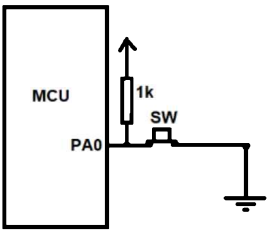
  PD12 = 0V

  PD[11:0] = 0V?

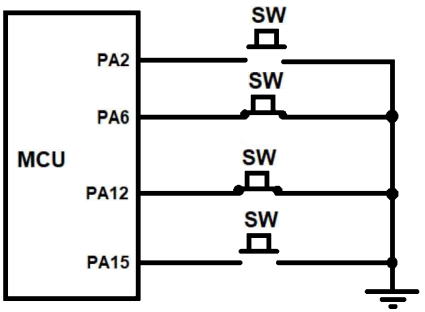
- Berapa nilai GPIOA_IDR, jika switch dilepas?



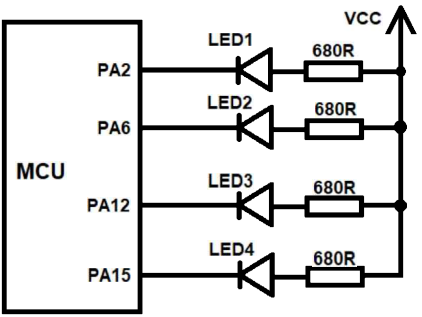
- Berapa nilai GPIOA_IDR, jika switch ditekan?



- Jika Internal *pull-up* resistor pada PORTA diaktifkan, maka berapa nilai GPIOA_IDR?

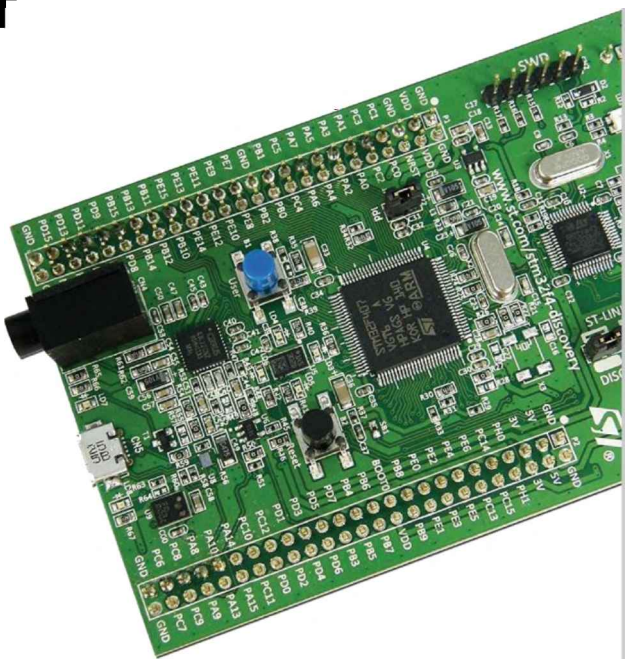


- Berapa nilai GPIOA_ODR agar LED1=ON, LED1=OFF, LED3=ON, LED4=OFF?



Sistem Embeded Berbasis ARM Cortex-M

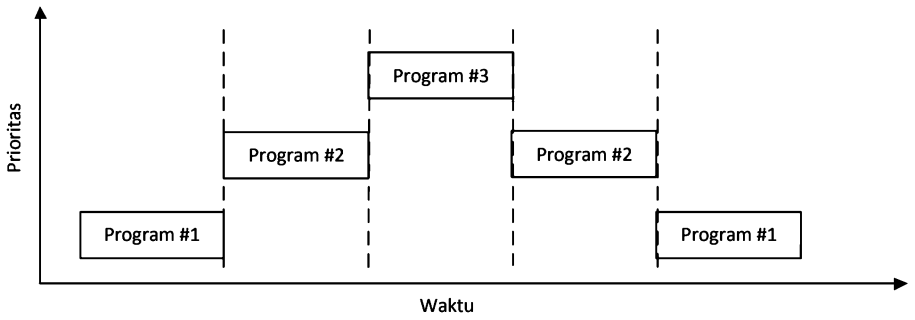
4



Bab 4 Interrupt

4.1. Konsep *Interrupt*

Interrupt (interupsi) merupakan fitur penting dari sebuah MCU. Fitur ini memungkinkan sebuah MCU meninggalkan program yang sedang dieksekusi untuk melakukan hal lain yang lebih prioritas. Setelah hal yang lebih prioritas itu selesai dikerjakan, maka MCU kembali melanjutkan eksekusi program reguler yang sedang dikerjakan sebelum terjadinya interupsi tadi. Gambar 18 menjelaskan interupsi secara sederhana. Program #1 sedang berlangsung ketika Program #2 dengan prioritas yang lebih tinggi melakukan *interrupt*. Di saat Program #2 sedang dieksekusi, Program #3 dengan prioritas tertinggi melakukan interupsi, sehingga proses pada Program #2 terhenti sementara. Setelah Program #3 selesai dieksekusi, maka MCU melanjutkan kembali Program #2. Dan setelah Program #2 selesai dieksekusi, maka MCU melanjutkan kembali proses pada Program #1.



Gambar 18. Ilustrasi *interrupt*

Interrupt pada STM32F4 dikontrol oleh *Nested Vectored Interrupt Controller* (NVIC). Setiap *external interrupt* memiliki beberapa *register* khusus yang berkaitan. Dalam reference manual STM32F4 dijabarkan tabel vektor pada NVIC ini. Silakan merujuk *ke reference*

manual. Sebagai contoh, STM32F405xx memiliki 88 *interrupt vector*[2], yang diberikan prioritas dari yang tertinggi sampai terendah.

Untuk menjalankan *interrupt*, kita dapat menggunakan fungsi tertentu yang dinamakan *interrupt handler*. Nama dari *interrupt handler* ini sudah ditentukan oleh CMSIS, sebagai contoh **USART1_IRQHandler(void)**.

4.2. External Interrupt/Event pada STM32F4

Pada *external interrupt* (EXTI), *interrupt* terjadi ketika nilai pin berubah. Perubahan dapat berupa:

1) *Rising edge*: *Interrupt* terjadi ketika pin berubah dari LOW ke HIGH
2) *Falling edge*: *Interrupt* terjadi ketika pin berubah dari HIGH to LOW
3) *Rising-Falling*: *Interrupt* terjadi ketika pin berubah, baik dari HIGH ke LOW maupun LOW ke HIGH

File header pada Keil terlihat sebagai berikut:

```
#define  GPIO_MODE_IT_RISING            0x10110000U
#define  GPIO_MODE_IT_FALLING           0x10210000U
#define  GPIO_MODE_IT_RISING_FALLING    0x10310000U
```

Rising trigger selection register (EXTI_RTSR) diset untuk mengatur *interrupt rising edge*. Nilai 0 berarti *interrupt rising edge* dinonaktifkan. Nilai 1 berarti *interrupt rising edge* diaktifkan. *Falling* trigger *selection register* (EXTI_FTSR) diset untuk mengatur *interrupt falling edge*. Nilai 0 berarti *interrupt falling edge* dinonaktifkan. Nilai 1 berarti *interrupt falling edge* diaktifkan. Perhatikan Gambar 19 dan Gambar 20.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

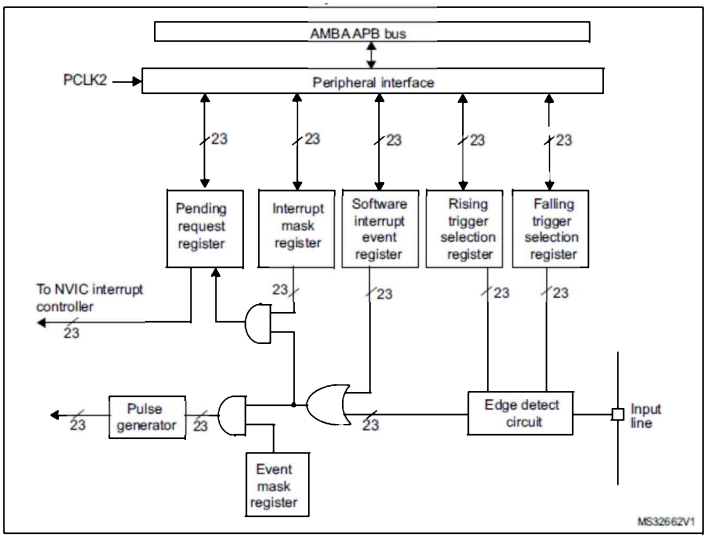
Gambar 19. EXTI_RTSR

[2] Beberapa *interrupt vector* masih belum digunakan (*reserved*)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 20. EXTI_FTSR

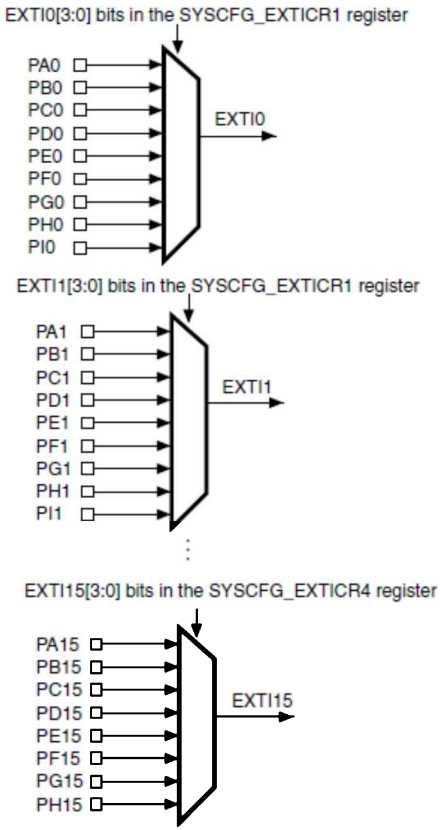
Pada dasarnya, semua *port* memiliki kemampuan EXTI. Untuk menggunakannya, *port* harus dikonfigurasi sebagai *input*. EXTI *Controller* terdiri atas 23 detektor untuk menghasilkan *interrupt/event request* (Gambar 21). Secara garis besar, *event* ini dibagi menjadi dua: 1) pin eksternal (P0 s.d P15), dan 2) *event* lainnya seperti RTC *interrupt*, Ethernet *interrupt*, USB *interrupt*, dan sebagainya.



Gambar 21. Diagram blok EXTI

Pada STM32F4, 140 GPIO terkoneksi dengan 16 *external interrupt/event* line (EXTI0 s.d EXTI15) sesuai dengan Gambar 22. Perhatikan bahwa GPIOx terkoneksi dengan *line* x, misalnya PA0 terkoneksi dengan *line* 0 (EXTI0), PB1 dengan (EXTI0), dan seterusnya. Dalam satu waktu hanya satu pin yang boleh terkoneksi

dalam *line* yang sama. Misalnya PA0 dan PC0 tidak boleh terkoneksi secara bersamaan karena sama-sama terkoneksi dengan *line* 0.



Gambar 22. Pemetaan GPIO ke 16 *external interrupt/event* pada STM32F405xx

STM32F4 memiliki 7 *interrupt handler* untuk pin GPIO, sebagaimana dijelaskan pada Tabel 9.

Tabel 9. *Interrupt handler* untuk pin GPIO

IRQ	Handler	Keterangan
EXTI0_IRQn	EXTI0_IRQHandler	Handler pin yang terkoneksi ke *line* 0

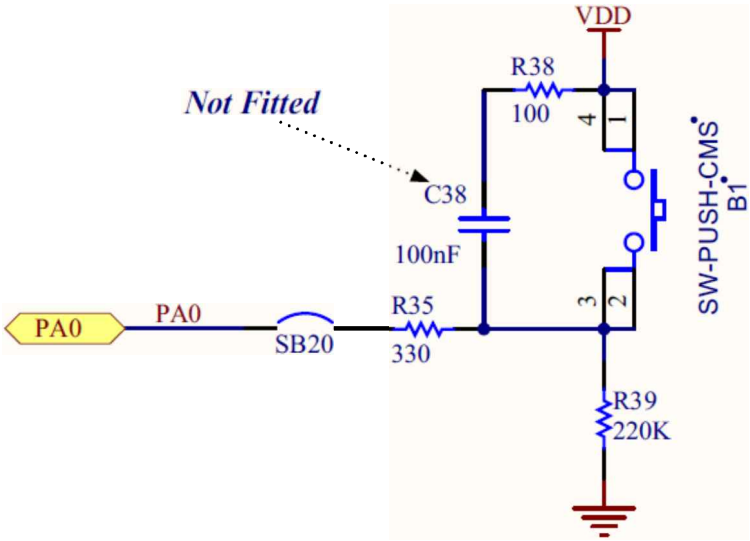
IRQ	Handler	Keterangan
EXTI1_IRQn	EXTI1_IRQHandler	Handler pin yang terkoneksi ke *line* 1
EXTI2_IRQn	EXTI2_IRQHandler	Handler pin yang terkoneksi ke *line* 2
EXTI3_IRQn	EXTI3_IRQHandler	Handler pin yang terkoneksi ke *line* 3
EXTI4_IRQn	EXTI4_IRQHandler	Handler pin yang terkoneksi ke *line* 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	Handler pin yang terkoneksi ke *line* 5 s.d 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	Handler pin yang terkoneksi ke *line* 10 s.d 15

4.2. Praktikum Bab 4

Alat dan Bahan

- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

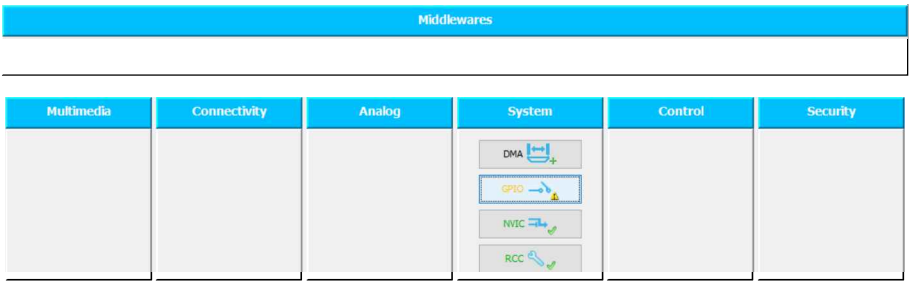
Pada praktikum ini, kita akan mendemonstrasikan EXTI menggunakan *push button* pada STM32F4DISCOVERY. *Push button* yang kita gunakan adalah B1. Koneksi B1 ini dapat kita lihat pada skematik di gambar di bawah. Ketika *button* ditekan, maka PA0 akan memiliki logika 1, sedangkan jika dilepas akan berlogika 0 (*pull down* menggunakan resistor R39 sehingga *pull down* internal tidak diperlukan lagi).

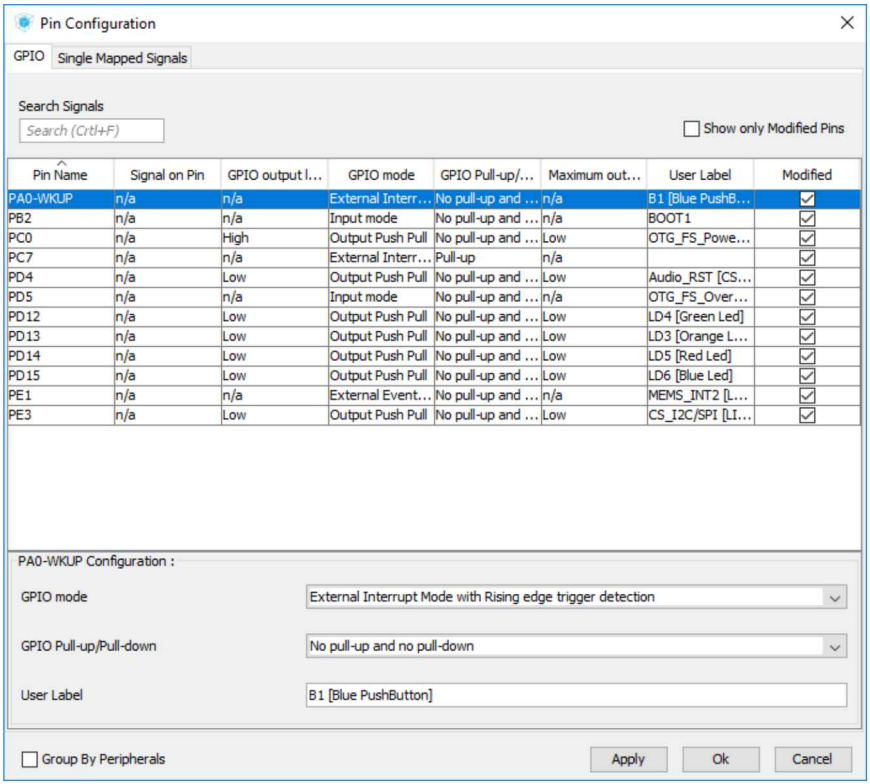


Praktikum 4-01

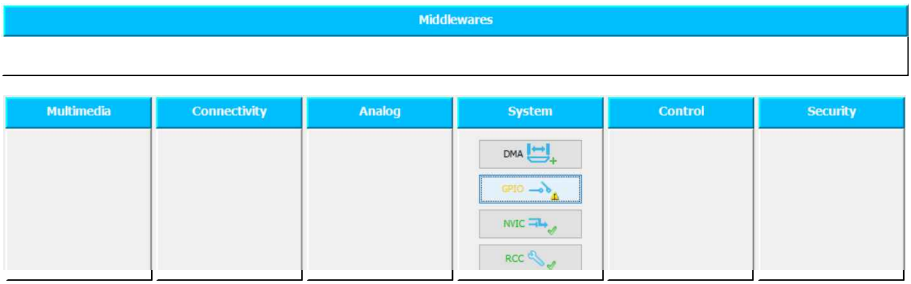
Tujuan program adalah men-*toggle* LED hijau ketika B1 ditekan (tegangan berubah dari LOW ke HIGH (*rising edge*)). Lakukan langkah-langkah seperti pada praktikum sebelumnya, namun ada perbedaan sebagai berikut:

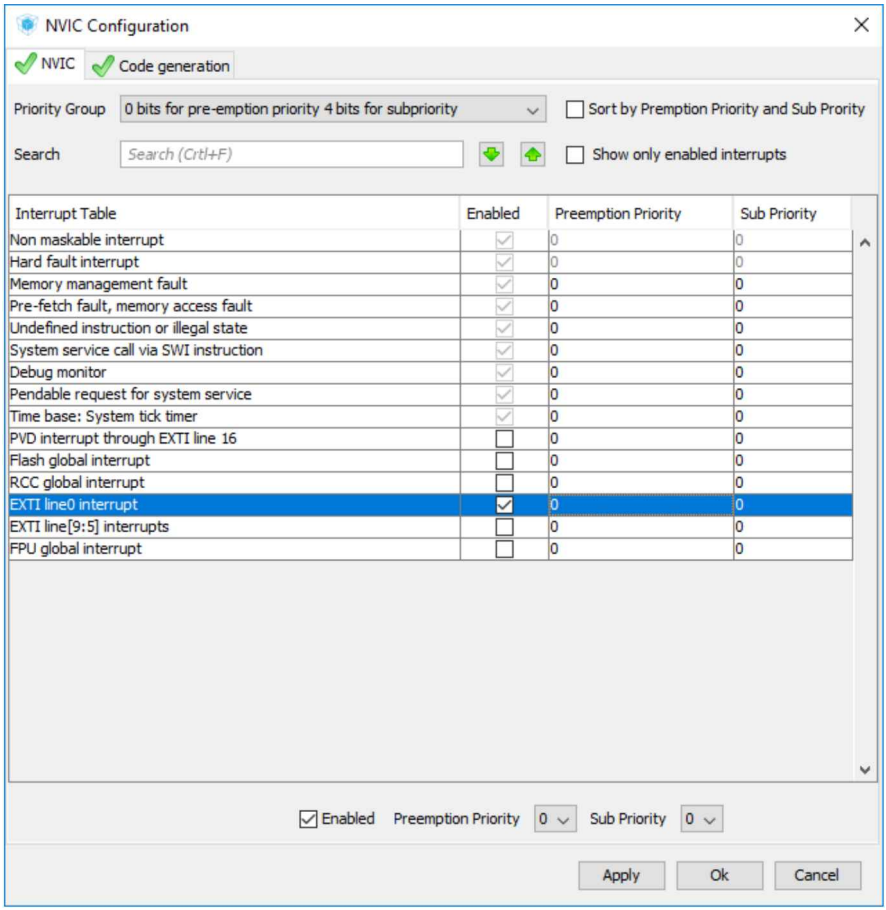
- Pada Configuration, klik GPIO dan set PA0 (B1) sebagai External Interrupt Mode with Rising edge trigger detection.





- Pada Configuration, klik NVIC dan check list EXTI line0 interrupt. Ini akan mengaktifkan EXTI channel 0 yang dapat digunakan oleh pin Px.0.





- Generate *code*, buka Keil, kemudian *build* program.
- Buka *file* stm32f4xx_it.c, dan temukan fungsi void EXTI0_IRQHandler(void). Masukkan kode sebagai berikut:

```
int a=0;
void EXTI0_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0); // Wajib. Otomatis
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12); // User's code
  a++; // User's code
}
```

- *Build* dan *upload* program. Coba tekan tombol B1 kemudian lepaskan. Lakukan secara berulang-ulang. Perhatikan yang terjadi pada LD4 (LED hijau) dan pahami.

Praktikum 4-02

Lakukan langkah-langkah pada praktikum 4-01, kemudian buka **main.c** dan temukan kode berikut di dalam fungsi static **void MX_GPIO_Init(void):**

```
/*Configure GPIO pin : B1_Pin */
GPIO_InitStruct.Pin = B1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
```

- Ganti Mode GPIO_MODE_IT_RISING menjadi GPIO_MODE_IT_FALLING. *Build* dan *upload* program. Coba tekan tombol B1 kemudian lepaskan. Lakukan secara berulang-ulang. Perhatikan yang terjadi pada LD4 (LED hijau). Apa perbedaannya dengan Praktikum 4-01?

Praktikum 4-03

Lakukan langkah praktikum 4-01, kemudian ubah mode **GPIO_MODE_IT_RISING** menjadi **GPIO_MODE_IT_RISING_FALLING**. *Build* dan *upload* program. Coba tekan tombol B1 kemudian lepaskan. Lakukan secara berulang-ulang. Perhatikan yang terjadi pada LD4 (LED hijau). Apa perbedaannya dengan Praktikum 4-01 dan 4-02?

4.3. Tugas Praktik Bab 4

Tugas Praktik 4-01

Tugas praktik ini menggunakan PC7 sebagai sumber *external interrupt*. Lakukan demonstrasi yang sama dengan praktikum 401, namun dengan ketentuan sebagai berikut:

- *Input* menggunakan PC7, bukan Button biru. Hubungkan-Lepaskan PC7 ke *ground* menggunakan *jumper* atau konduktor lain untuk menguji program (sebagai pengganti button).
- Mode *interrupt*: FALLING_EDGE.
- PC7 diset *pull-up*, sehingga bernilai HIGH jika tidak terkoneksi.
- Lampu yang digunakan adalah LED biru.

Tugas Praktik 4-02

Tugas praktik ini menggunakan *push button* untuk *toggle* pergerakan LED. Nyalakan LED3, LED4, LED5, dan LED6 dengan berurutan setiap detik sesuai dengan Tabel 10. Gunakan *push button* B1 dengan mode *interrupt* (pilih salah satu: *Falling Edge* atau *Rising Edge*). Jika B1 ditekan (kemudian dilepas), LED menyala dengan urutan seperti Tabel 11. Namun, jika B1 ditekan sekali lagi (kemudian dilepas), LED menyala berurutan setiap detik dengan arah sebaliknya, yakni seperti tabel di bawah. Demikian seterusnya bergantian antara Tabel 10 dan Tabel 11 (*toggled*).

Tabel 10

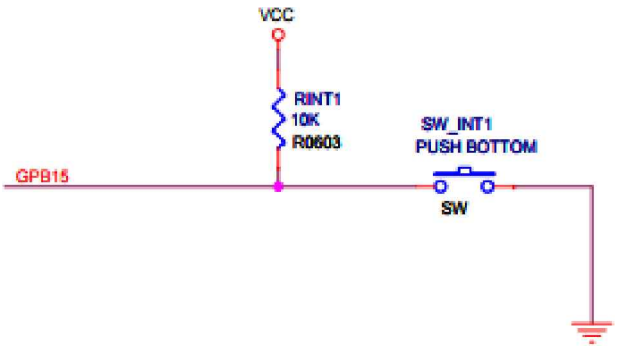
Detik ke	LED Hijau	LED Jingga	LED Merah	LED Biru
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON
5	ON	OFF	OFF	OFF
dan seterusnya	...	...	...	...

Tabel 11

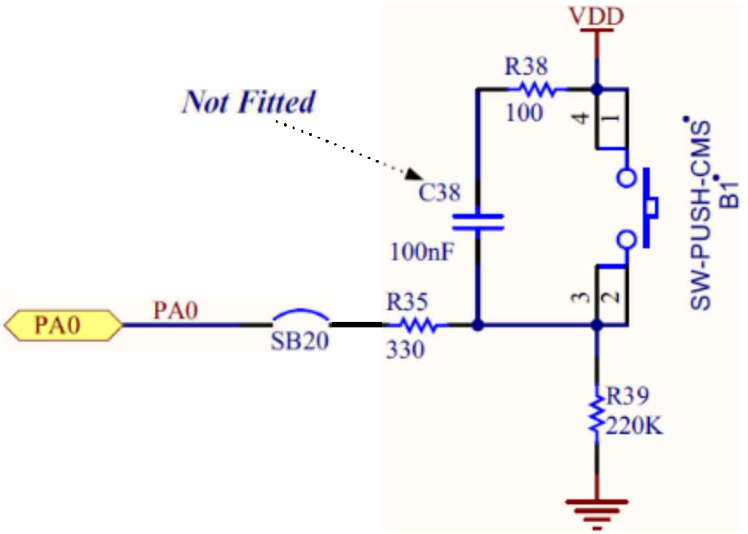
Detik ke	LED Hijau	LED Jingga	LED Merah	LED Biru
1	OFF	OFF	OFF	ON
2	OFF	OFF	ON	OFF
3	OFF	ON	OFF	OFF
4	ON	OFF	OFF	OFF
5	OFF	OFF	OFF	ON
dan seterusnya	...	...	...	...

4.4. Latihan Soal Bab 4

- Jelaskan perbedaan antara *rising edge* dan *falling edge*!
- Jika digunakan *external interrupt* mode *rising edge*, maka *interrupt* terjadi ketika tombol ditekan atau dilepas?

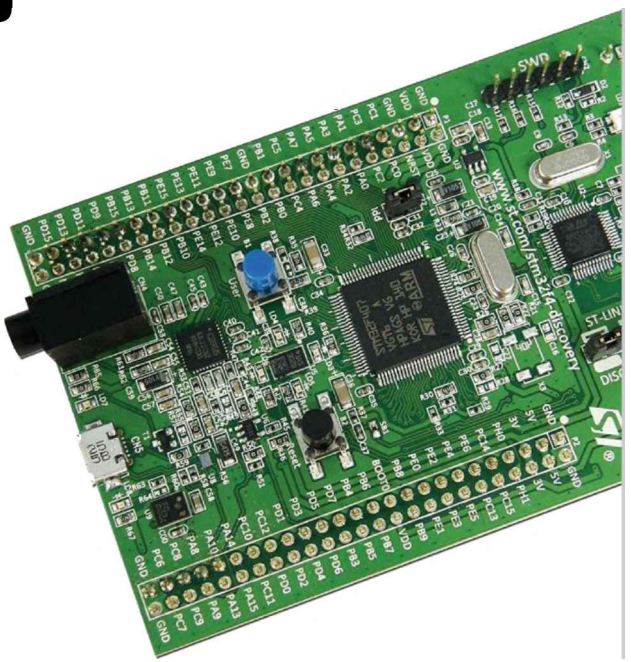


- Jika digunakan *external interrupt* mode *falling edge*, maka *interrupt* terjadi ketika tomboditekan/dilepas?



Sistem Embeded Berbasis ARM Cortex-M

5



Bab 5 USART

5.1. Prinsip Kerja USART

Universal Synchronous Asynchronous Receiver Transmitter (USART) pada STM32F4 mendukung komunikasi antar mikroprosesor atau komponen lain dengan mode *full-duplex* maupun *half-duplex* sesuai standar industri, baik sinkron maupun asinkron. Selain itu, USART ini mendukung juga komunikasi *Local Interconnction Network* (LIN), protocol Smartcard, Infrared Data Association (IrDA), dan modem (CTS/RTS).

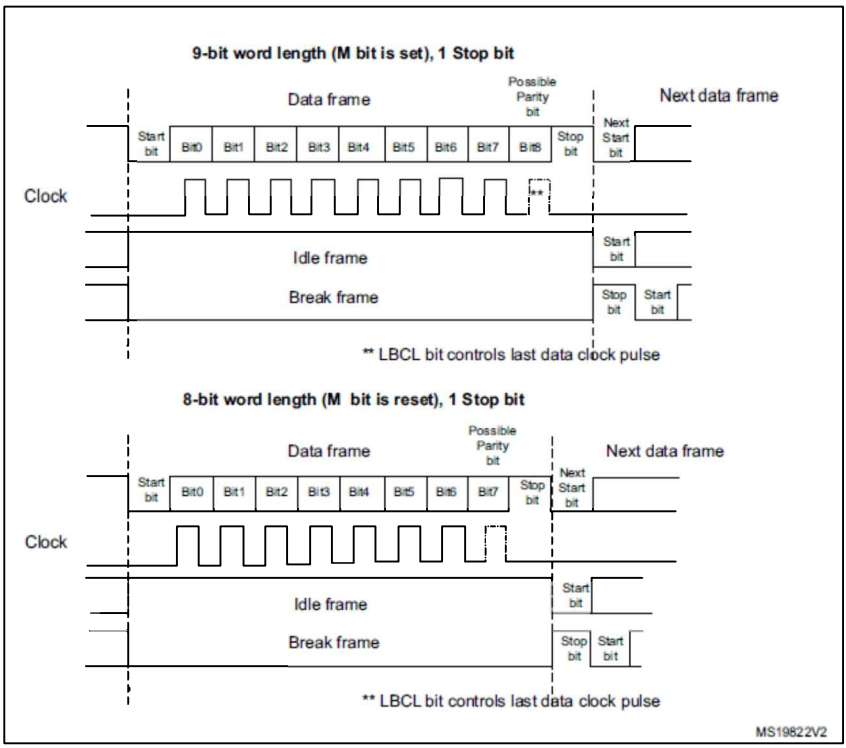
Fitur utama USART yang dimiliki STM32F4 dapat dijabarkan sebagai berikut:

- Komunikasi *full-duplex*/*half-duplex*, baik sinkron maupun asinkron
- Metode *oversampling* yang dapat dikonfigurasi, yakni 16 atau 8
- Bit data: 8 atau 9 bit
- Bit stop: 0,5, 1, 1,5 atau 2 bit
- Flow *Control*: CTS/RTS
- *Output clock* untuk komunikasi sinkron
- Dapat menggunakan *Direct Access Memory* (DMA)
- Flag yang dapat dibaca: Receive buffer full, Transmit buffer empty, End of transmission
- *Parity*: Genap, ganjil, atau *none*
- Deteksi *Error*: Overrun *Error*, Noise detection, Frame *Error*, Parity *Error*
- Memiliki 10 jenis *interrupt* yang berkaitan (lihat Tabel 12):

Tabel 12. USART *interrupt* requests

Interrupt event	Event flag	Enable control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NF or ORE or FE	EIE

Gambar 23 mendeskripsikan diagram waktu dari USART. Pin TX (*transmitter*) berada dalam keadaan *low* saat *start bit* dan dalam keadaan *high* saat *stop* bit dan *idle*.

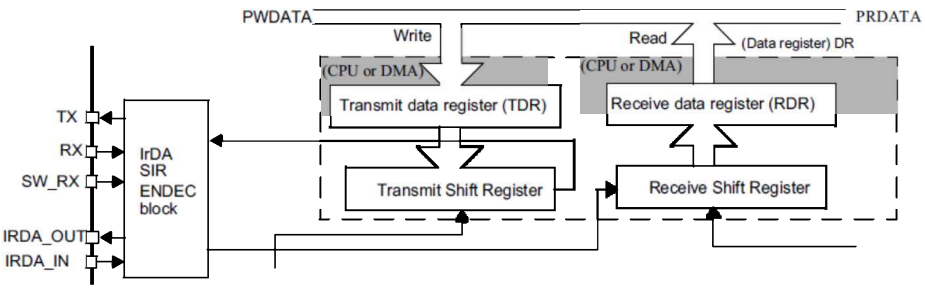


Gambar 23. Diagram waktu USART

STM32F4 memiliki 6 kanal USART yang dinamakan: USART1, USART2, USART3, UART4, UART5, dan USART6. Namun demikian, fitur yang mereka memiliki berbeda-beda sebagaimana dijelaskan pada Tabel 13.

Tabel 13. Konfigurasi USART

USART modes	USART 1	USART 2	USART 3	UART4	UART5	USART 6
Asynchronous mode	X	X	X	X	X	X
Hardware flow control	X	X	X	NA	NA	X
Multibuffer communication (DMA)	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous	X	X	X	NA	NA	X
Smartcard	X	X	X	NA	NA	X
Half-duplex (single-wire mode)	X	X	X	X	X	X
IrDA	X	X	X	X	X	X
LIN	X	X	X	X	X	X



Gambar 24. Diagram blok USART

Ada beberapa *register* yang berhubungan langsung dengan USART, di antaranya *status register* (USART_SR), *data register* (USART_DR), *baudrate register* (USART_BRR), *Control register* 1 (USART_CR1), *Control register* 2 (USART_CR2), *Control register* 3 (USART_CR3), dan *Guard time and prescaler register* (USART_GTPR). *Register-register* tersebut dijelaskan dalam Tabel 14.

Tabel 14. *Register-register* yang berhubungan langsung dengan USART

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																						DR[8:0]									
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																				ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																GT[7:0]												PSC[7:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Gambar 25. USART Status *Register* (USART_SR)

USART_SR digambarkan pada Gambar 25. Bit *transmit data register empty* (TXE) pada status *register* (USART_SR) menandakan apakah isi *register* TDR sudah selesai ditransfer ke shift *register*. TXE=1 jika isi TDR sudah selesai ditransfer ke shift *register* dan TXE=0 jika transfer data belum selesai. *Interrupt* akan terjadi jika bit TXEIE pada *register* USART_CR1 diset 1.

Bit *read* data *register* not empty (RXNE) menandakan apakah data pada *read data register* (RDR) sudah siap untuk dibaca,

misalnya dengan memindahkan nilai RDR ke dalam sebuah variabel. RXNE bernilai 1 jika data pada RDR sudah siap dibaca dan RXNE bernilai 0 jika data belum siap dibaca. *Interrupt* akan terjadi jika bit RXEIE pada *register* USART_CR1 (Gambar 26) diset 1.

Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Gambar 26. USART *Control register* 1 (USART_CR1)

Bit USART *enable* (UE) pada USART_CR1 berfungsi untuk mengaktifkan USART. Masukkan nilai UE=1 jika USART ingin diaktifkan dan UE=0 jika USART ingin dinonaktifkan. TXE *interrupt* enable (TXEIE) berfungsi untuk mengaktifkan *interrupt* jika TXE=1. Masukkan nilai TXEIE=1 jika ingin mengaktifkan intetrupt dan TXEIE=0 jika ingin menonaktifkan *interrupt*.

Komunikasi UART pada STM32F4 terdiri atas *polling* dan *interrupt*. Fungsi-fungsi yang berkaitan dapat dilihat pada **stm32f4xx_hal_uart.c**. Beberapa fungsi terkait dengan UART adalah sebagai berikut:

- Transmit *Polling*: HAL_UART_Transmit()
- Transmit *Interrupt*: HAL_UART_Transmit_IT()
- Receive *Polling*: HAL_UART_Receive()
- Receive *Interrupt*: HAL_UART_Receive_IT()

5.1.1. Mengirim UART dengan *polling*

Fungsi pada kode berikut ini digunakan untuk mengirim UART dengan *polling*, di mana *huart adalah nama UART yang dipakai, *pData adalah *point*er dari data yang akan dikirimkan (data buffer), Size adalah ukuran data, Timeout adalah durasi *timeout* (dalam ms).

```
HAL_UART_Transmit(UART_HandleTypeDef
                  *huart,
                  uint8_t *pData,
                  uint16_t Size,
                  uint32_t Timeout)
```

5.1.2. Menerima UART dengan *polling*

Fungsi pada kode berikut ini digunakan untuk menerima UART dengan *polling*, di mana *huart adalah nama UART yang dipakai, *pData adalah *point*er dari data yang diterima (*data buffer*), Size adalah ukuran data, Timeout adalah durasi *timeout* (dalam ms).

```
HAL_UART_Receive(UART_HandleTypeDef *huart,
                 uint8_t *pData,
                 uint16_t Size,
                 uint32_t Timeout)
```

5.1.3. Mengirim UART dengan *interrupt*

Fungsi pada kode berikut ini digunakan untuk mengirim UART dengan *interrupt*. Selain fungsi tersebut, kita perlu membuat fungsi baru yang bernama HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart), yang dibuat secara manual. Fungsi ini merupakan *interrupt handler* yang akan dipanggi setiap selesai pengiriman UART.

```
HAL_UART_Transmit_IT(UART_HandleTypeDef *huart,
                     uint8_t *pData,
                     uint16_t Size)
```

5.1.4. Menerima UART dengan *interrupt*

Fungsi pada kode berikut ini digunakan untuk menerima UART dengan *interrupt*. Selain fungsi tersebut, kita perlu membuat fungsi baru yang bernama HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart), yang dibuat secara manual. Fungsi ini merupakan *interrupt handler* yang akan dipanggi setiap selesai proses penerimaan UART.

```
HAL_UART_Receive_IT(UART_HandleTypeDef *huart
                    uint8_t *pData
                    uint16_t Size)
```

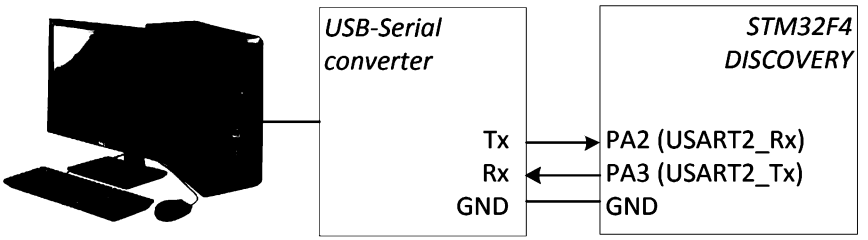
5.2. Praktikum Bab 5

Alat dan Bahan

- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)
- Opsional: Modul USB to Serial *Converter* (Gambar 27) (1 unit)
- Kabel *jumper* (secukupnya)

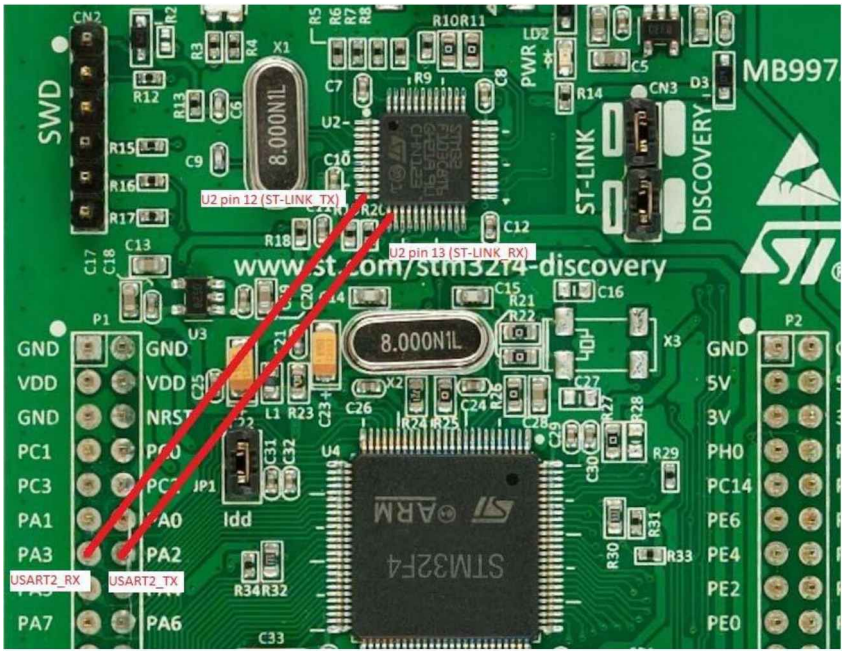
Gambar 27. USB-UART *Converter*

Kita dapat menggunakan USB-to-UART *Converter* untuk melakukan komunikasi serial. Hubungkan sistem minimum STM32F4DISCOVERY dengan modul USB-to-Serial *Converter* seperti pada Gambar 28. Jangan lupa untuk menginstalasi *driver* USB-UART *Converter* sebelum digunakan.



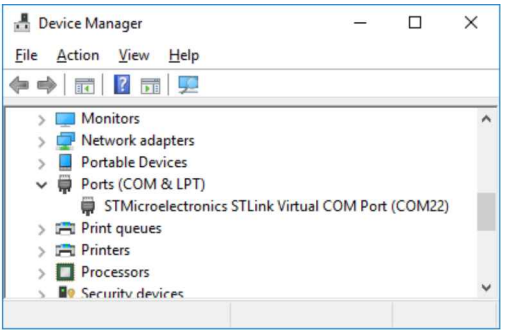
Gambar 28. Koneksi *Hardware* pada praktikum UART

Pilihan lain untuk melakukan komunikasi serial adalah memodifikasi STM32F4DISCOVERY. Pada dasarnya, STM32F4DISCOVERY memiliki USB-to-UART *Converter*, yakni ST-LINK/V2-A virtual COM *port* (VCP). Namun diperlukan koneksi tambahan dengan cara menghubungkan U2 pin 12 *and* 13 dengan STM32F407 USART2 (PA2 dan PA3: pin 14 dan 13 pada konektor P1) sebagaimana terlihat pada Gambar 29. Dengan demikian, USB-to-UART *Converter* seperti pada Gambar 27 dan Gambar 28 tidak diperlukan lagi.



Gambar 29. Koneksi ST-LINK VCP dengan USART2 pada STM32F407

Praktikum pada buku ini menggunakan VCP dengan cara memodifikasi STM32F4DISCOVERY. Jika sudah terkoneksi, maka pada *device manager* akan muncul STLink Virtual COM Port pada *device manager* Windows (Gambar 30).



Gambar 30. STLink Virtual COM *Port* terdeteksi pada komputer

Praktikum 5-01

(501-a)

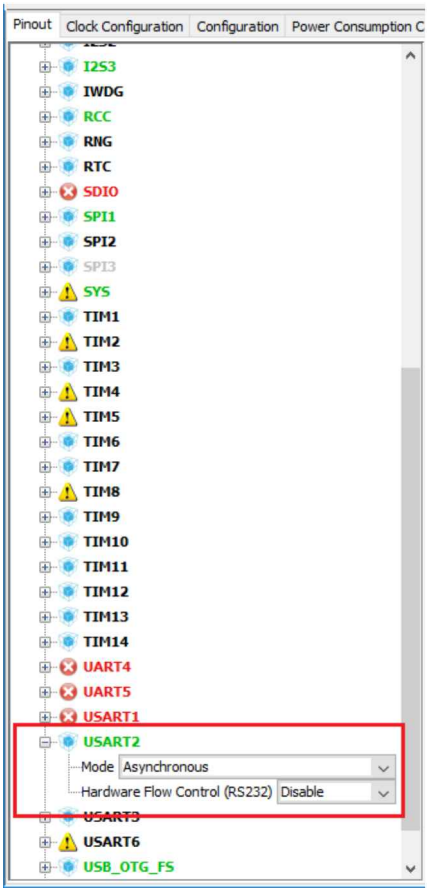
Pada praktikum kali ini, akan dilakukan uji coba program untuk mengirimkan data serial UART dengan sistem *polling*. Periferal UART yang digunakan adalah UART2 yang terhubung dengan pin PA2 (*transmitter*) dan PA3 (*receiver*). Data akan dikirimkan ke komputer melalui VCP Parameter UART yang digunakan pada percobaan kali ini adalah sebagai berikut:

- *Baudrate*: 115200
- Bit data: 8
- Parity: none
- Bit Stop bit: 1

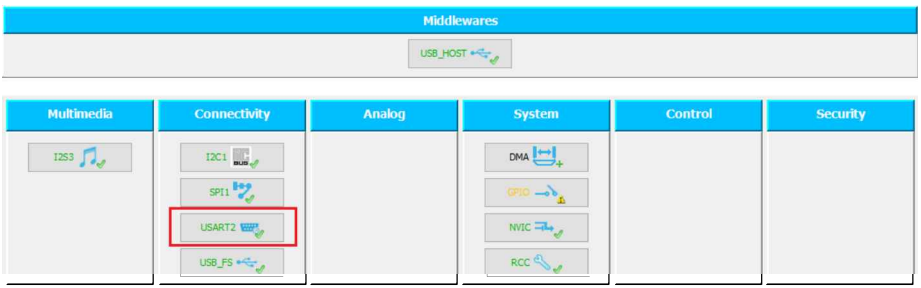
Data yang akan dikirimkan ke komputer merupakan data byte dan ASCII dari teks "Hello World from UART" dan diakhiri dengan karakter ENTER (CR+LF). Jeda tiap pengiriman data adalah 500 ms agar data yang diterima di komputer dapat diamati dengan baik. Lakukan langkah-langkah berikut:

- Buat *project* baru menggunakan STM32CubeMX dengan memilih STM32F4DISCOVERY pada menu Board Selector. Aktifkan *check box* Initialize all peripherals with their default mode agar STM32CubeMX melakukan konfigurasi standar terhadap periferal dan distribusi *clock* sesuai dengan konfigurasi STM32F4DISCOVERY.

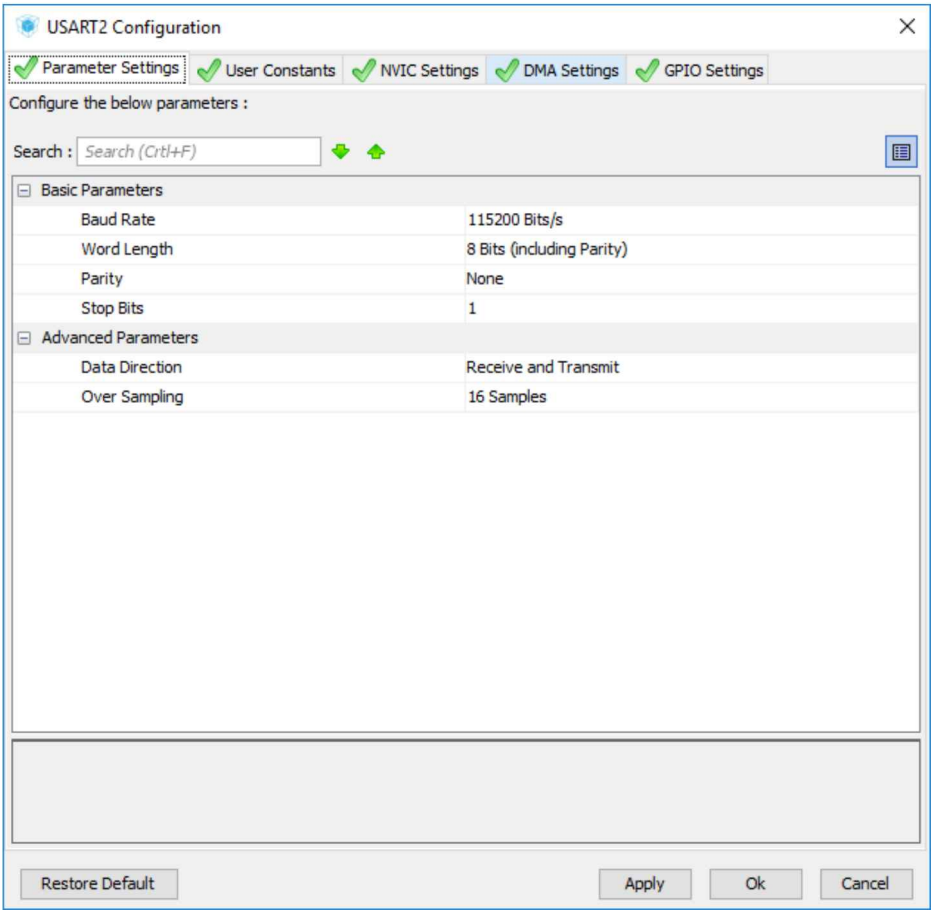
- Lakukan konfigurasi pada menu Pinout dengan mengaktifkan periferal USART2 dengan mode Asynchronous.



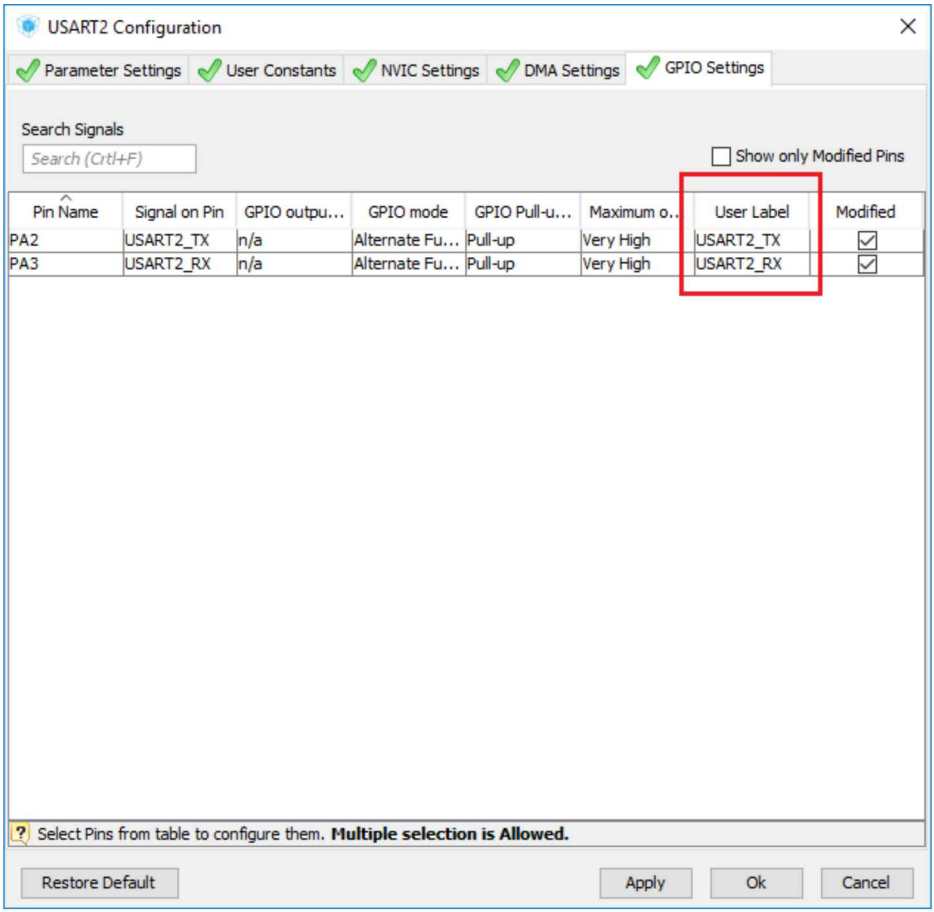
- Lakukan konfigurasi pada menu Configuration dengan memilih USART2 pada bagian Connectivity.



- Pada menu Parameter Setting Anda dapat melakukan pengaturan *baudrate*, ukuran *data bits*, *parity*, *stop bits* dan parameter komunikasi serial lainnya.



- Pada menu GPIO Settings ubahlah nama pin pada bagian User Label sesuai dengan nama yang diinginkan. Langkah ini sifatnya tidak wajib, namun dapat memudahkan dalam mengingat nama pin-pin yang digunakan ketika membuat program.

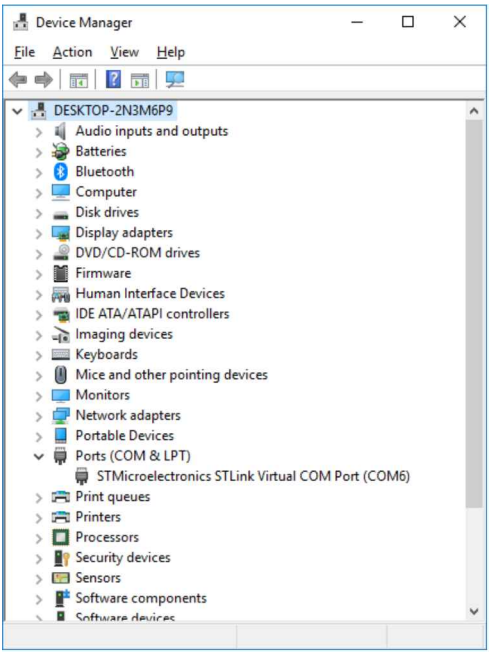


- Generate *project* dan buka Keil. Buka *project* menggunakan Keil IDE. Kemudian tambahkan program pada *file* main.c seperti berikut ini:

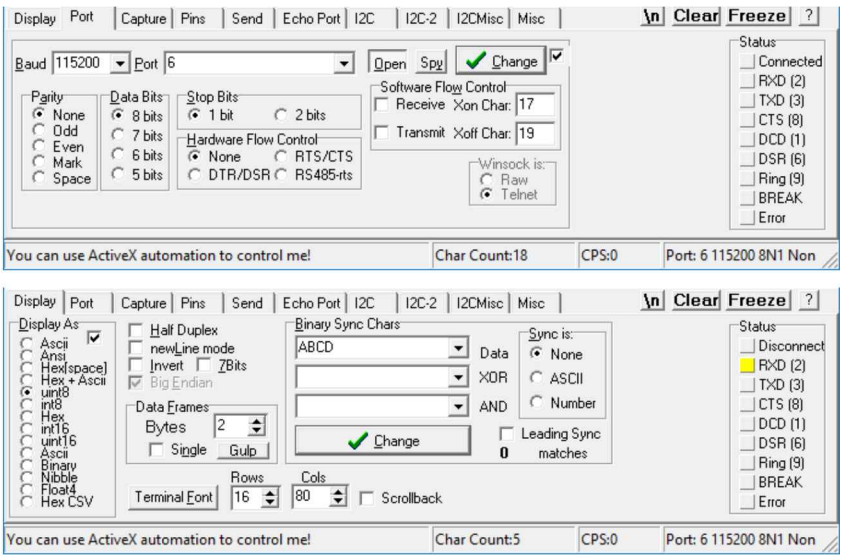
```
uint8_t data = 0;
while(1)
{
 HAL_UART_Transmit(&huart2,(uint8_t*)&data,1,10);
 HAL_Delay(500);
 data++;
}
```

- *Build* dan *load* program.
- Lakukan uji coba program dengan cara menghubungkan kabel USB to Serial ke komputer. Kemudian cek Device Manager untuk

melihat nama *port* (COM) dari STLink. Perhatikan pada COM berapa STLink terkoneksi.



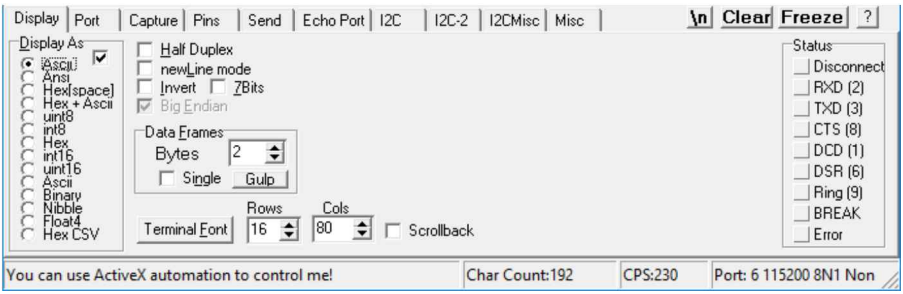
- Buka RealTerm dan pastikan nama COM sesuai, Baud = 115200, dan display = uint8. Klik Change setiap kali mengubah parameter.



- Analis data yang muncul pada RealTerm dan pahami.
- Setelah memahaminya, ubah program menjadi sebagai berikut:

```
char teks[] = "Hello World from UART\r\n";
while(1)
{

 HAL_UART_Transmit(&huart2,(uint8_t*)teks,strlen(teks),10);
 HAL_Delay(500);
}
```

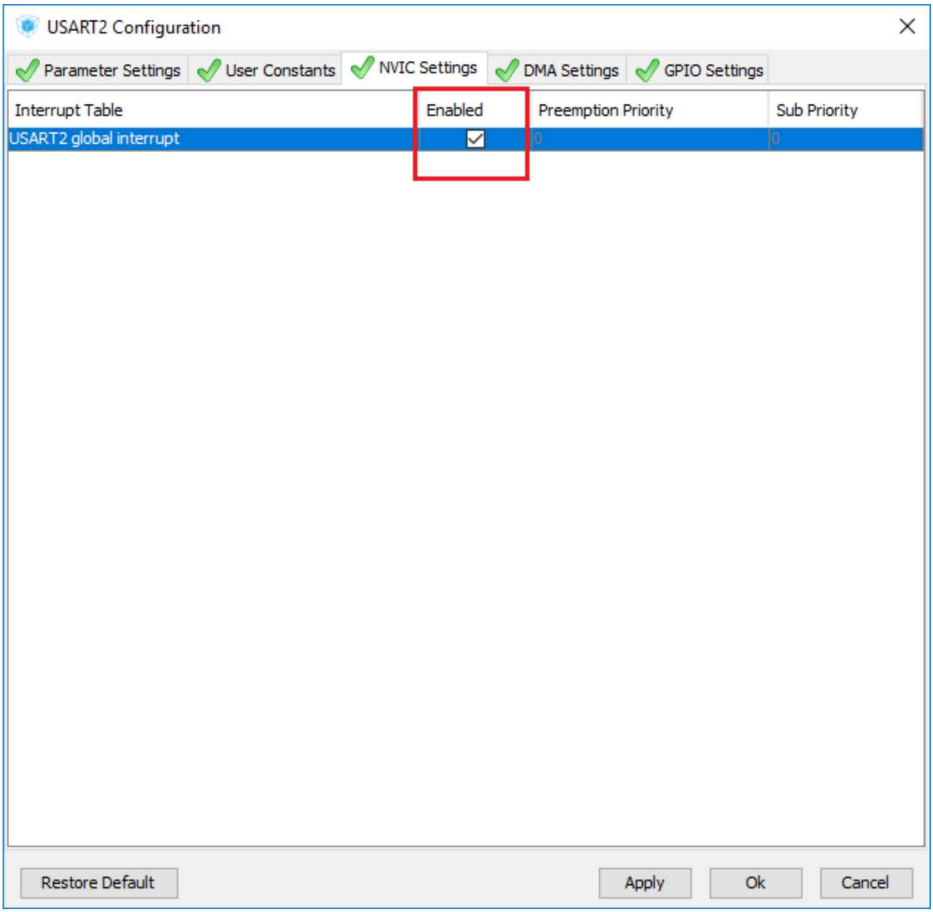
- Ubah display pada RealTerm menjadi Ascii.



- Analis data yang muncul pada RealTerm dan pahami.

(501-b)

- Buat *project* baru dengan konfigurasi seperti percobaan sebelumnya. Pada pengaturan USART Configuration aktifkan USART2 global interrupt pada *tab* NVIC Settings. Ini bertujuan untuk mengaktifkan interupsi pada periferal USART.



- Generate kode dan buka Keil.
- Pada main.c tambahkan *header file* string.h

```
#include "string.h"
```

- Tambahkan fungsi *callback* di bawah ini pada *file* main.c di lokasi sebelum fungsi main. Fungsi *callback* merupakan fungsi yang akan dieksekusi apabila terjadi interupsi yang bersumber dari USART. Pada percobaan kali ini interupsi akan terjadi jika proses pengiriman data telah komplit. Fungsi *callback* akan tetap dieksekusi apabila terjadi interupsi, walupun tidak pernah dipanggil pada fungsi main.

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
 UNUSED(huart);
 HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
}
```

- Lakukan modifikasi fungsi main menjadi seperti di bawah ini:

```
char *data = "Hello World From USART - Interrupt Transmit Mode \r\n";
while (1)
{
 HAL_UART_Transmit_IT(&huart2,(uint8_t *)data,strlen(data));
 HAL_Delay(500);
}
```

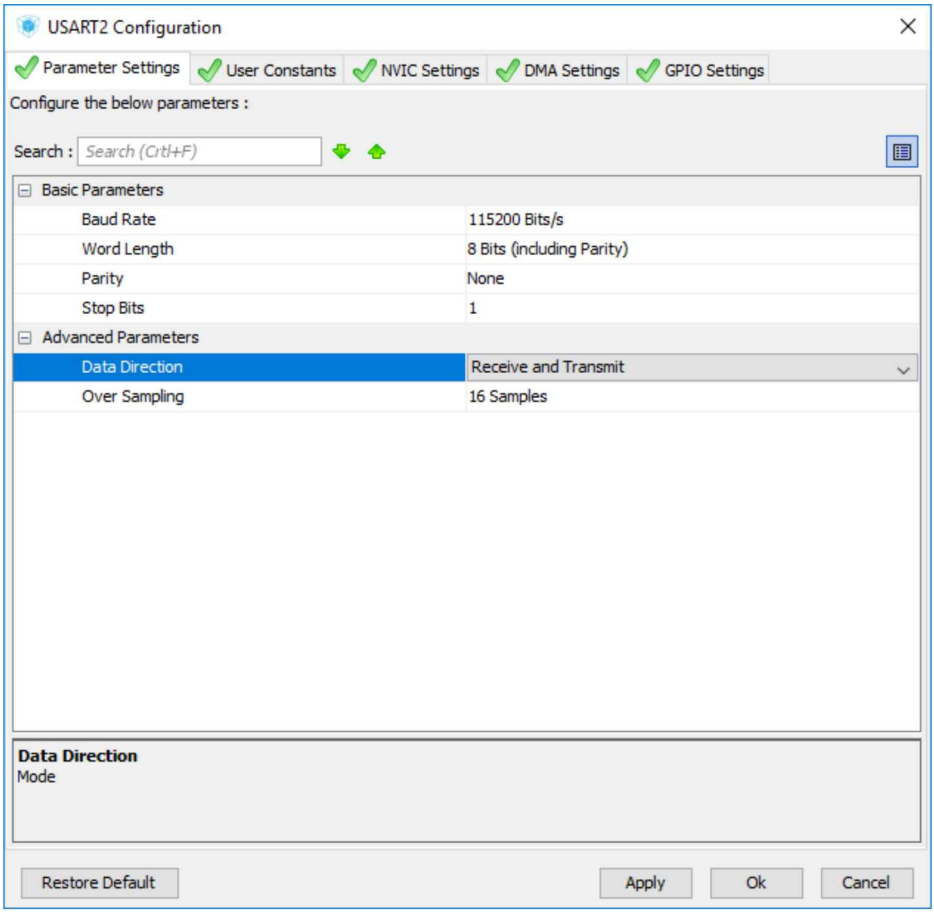
- *Build* dan *load* program ke mikrokontroler. Buka RealTerm (display=Ascii) dan perhatikan data yang dikirimkan oleh mikrokontroler. Perhatikan juga nyala LED3 pada STM32F4DISCOVERY. LED3 akan berkedip jika transmisi data telah komplit atau telah selesai.

Praktikum 5-02

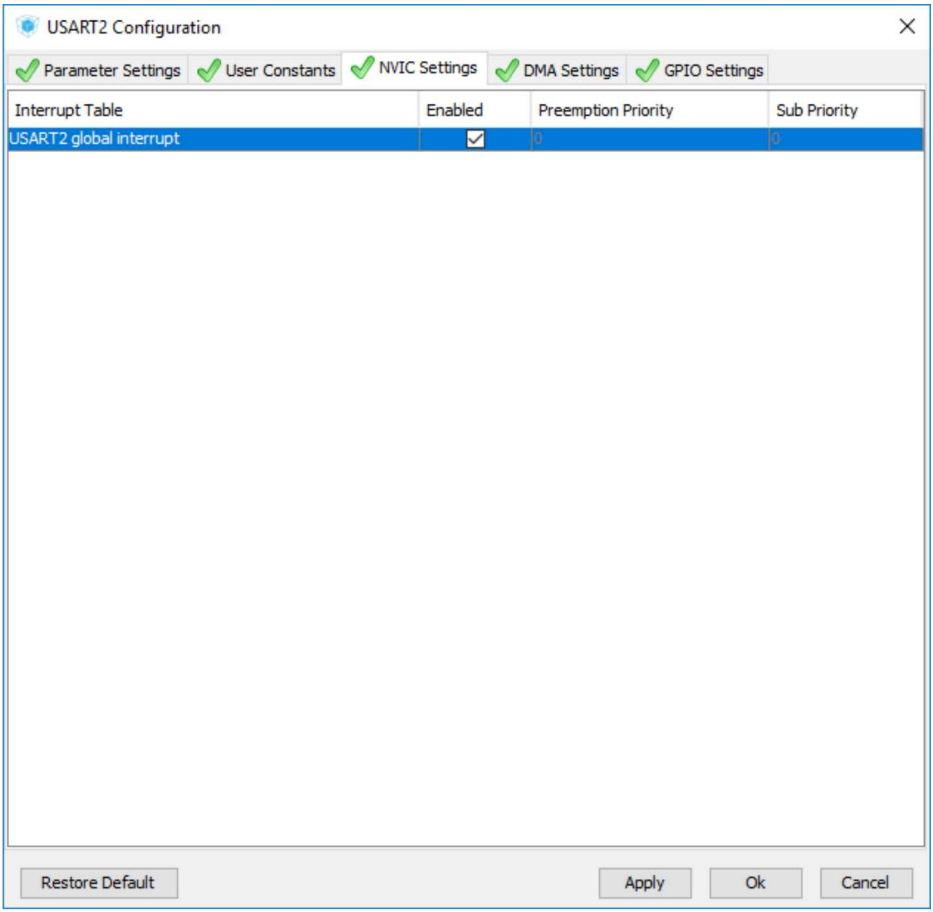
Pada program ini, kita akan mengubah status LED orange (*toggle*) setiap kali STM32F407 menerima 1 byte data dari PC (USART *Receive Interrupt*). Lakukan langkah-langkah berikut:

- Buat program baru menggunakan STM32CubeMX. Set USART2 menjadi Asynchronous.

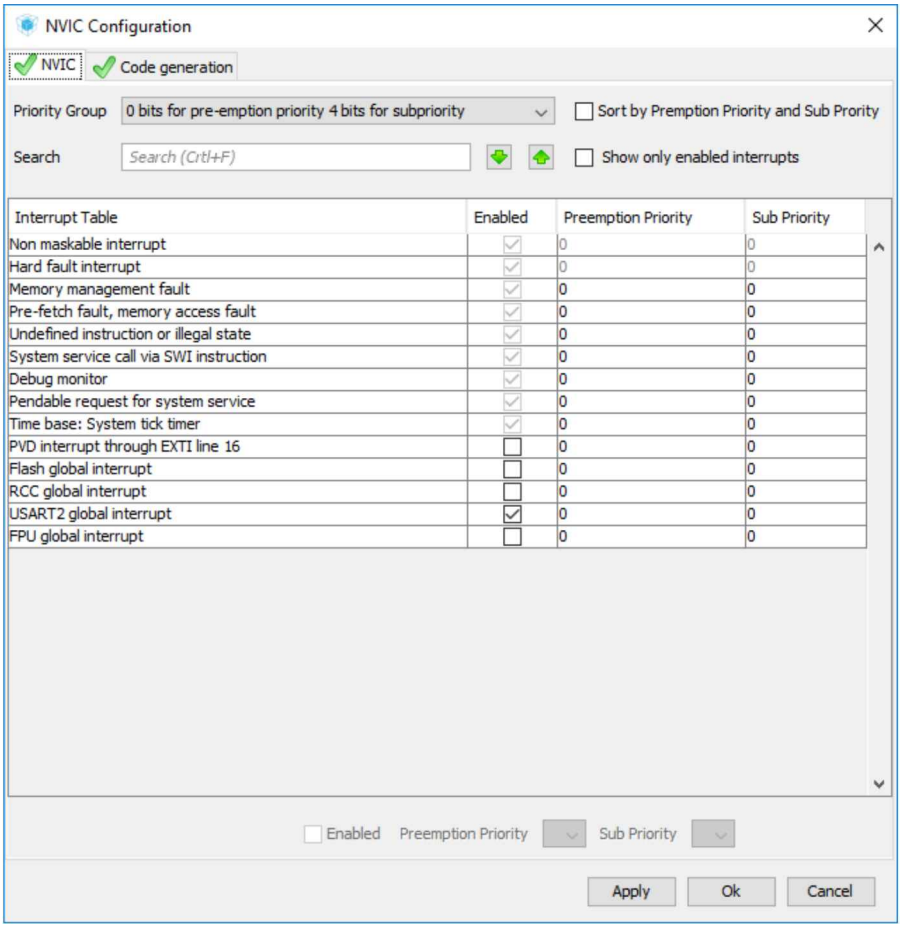
- Pada USART Configuration, set parameter sebagai berikut:



- Pada *tab* NVIC Settings aktifkan USART2 global interrupt.



- Pada NVIC Configuration, pastikan USART2 global interrupt sudah aktif.



- *Generate* code dan buka Keil.
- Tambahkan kode berikut sebelum fungsi main():

```
uint8_t a = 0;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
 UNUSED(huart);
 HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
 HAL_UART_Receive_IT(&huart2, (uint8_t*)&a, 1);
}
```

- Masukkan kode berikut di dalam fungsi main():

```
HAL_UART_Receive_IT(&huart2, (uint8_t *)&a, 1);
while (1)
{

}
```

- *Build* dan load program.
- Buka RealTerm dan pastikan COM sesuai, baudrate = 115200. Buka *tab* Send dan masukkan satu angka sebarang (pilih 0 s.d 9) dan tekan tombol Send Number. Perhatikan perubahan pada LED orange.
- Pahami kode program tersebut.

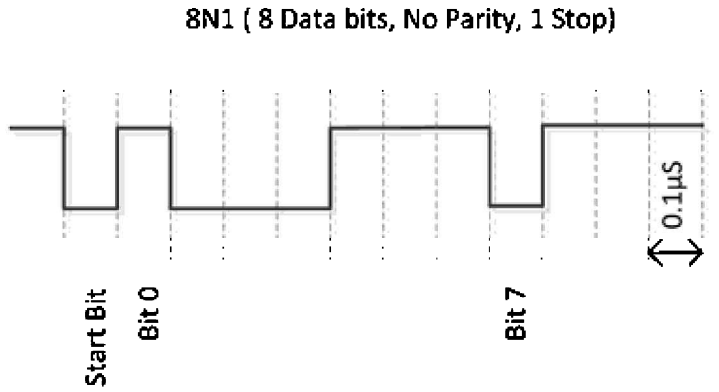
5.3. Tugas Praktik Bab 5

Buat sebuah program yang bertujuan untuk menyalakan LED sesuai angka yang dimasukkan. Angka dimasukkan melalui RealTerm. Perhatikan tabel di bawah.

Angka dimasukkan	LED Menyala
1	*Blue*
2	*Red*
3	*Orange*
4	*Green*
5	*Blue* dan *Red*
6	*Orange* dan *Green*
7	Semua

5.4. Latihan Soal Bab 5

- Apa data yang dikirimkan?



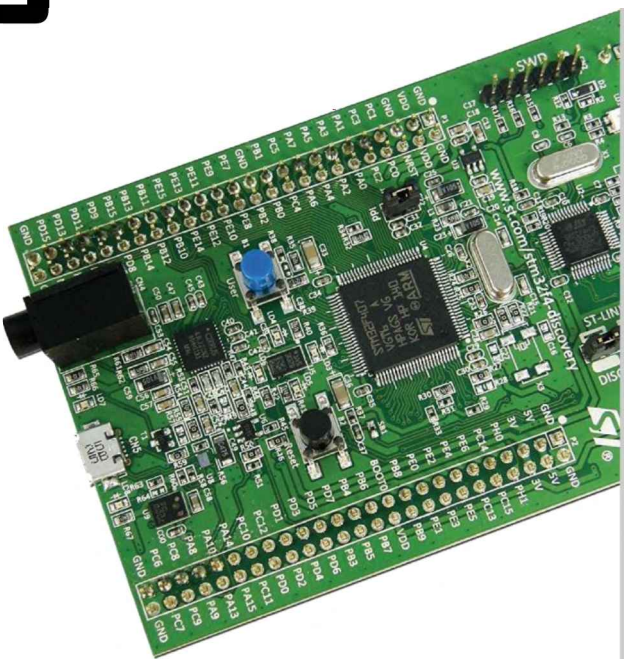
- Jika program di bawah dijalankan, apa yang akan muncul pada RealTerm (Ascii)?

```
char *data = "Hello World From USART - Interrupt \r\n";
while (1)
{
        HAL_UART_Transmit_IT(&huart2,(uint8_t *)data,7);
        HAL_Delay(500);
}
```

- Pada UART 9600 8N1 (baudrate 9600, No parity, 1 stop bit), berapa waktu tercepat yang diperlukan untuk mengirim 48 buah karakter ASCII?

Sistem Embeded Berbasis ARM Cortex-M

6



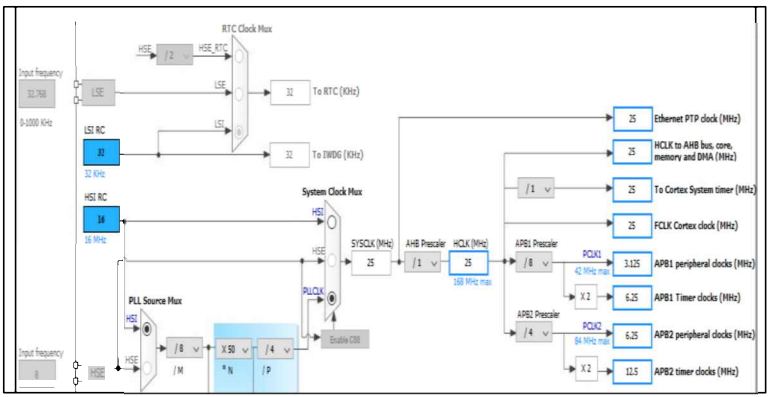
Bab 6 Timer/Counter

6.1. Prinsip Kerja *Timer/Counter*

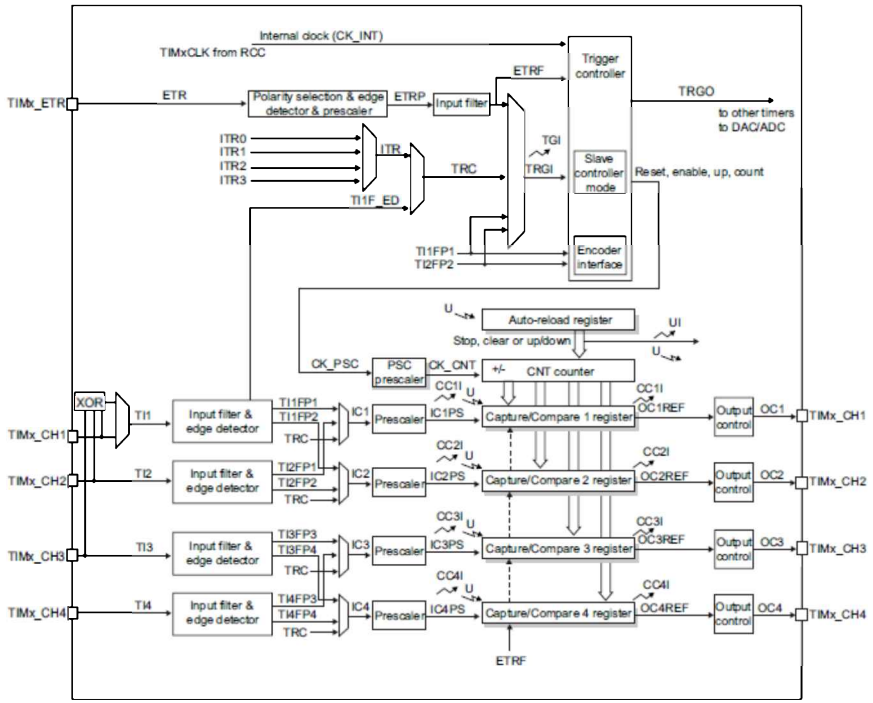
Timer/Counter adalah fitur yang dimiliki kebanyakan mikrokontroler untuk menentukan waktu dengan periode tertentu dengan memanfaatkan sistem *clock* yang dimilikinya. *Timer* pada STM32 dapat diklasifikasikan ke dalam kategori sebagai berikut:

1) *Advanced-control timer*s seperti TIM1 dan TIM8. Keduanya merupakan *timer* 16-bit.
2) *General-purpose configuration timer*s seperti TIM2, TIM3, TIM4, TIM5, TIM9, TIM10, TIM11, TIM12, TIM13, dan TIM14. TIM2 dan TIM5 merupakan *timer* 32-bit, sedangkan yang lainnya merupakan *timer* 16-bit.
3) *Basic-configuration timer*s seperti TIM6 *and* TIM7. Keduanya merupakan *timer* 16-bit.

Perhatikan Gambar 31. TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM12, TIM13, dan TIM14 diatur oleh APB1 *timer clock*, sedangkan TIM1, TIM8, TIM11, TIM10, TIM9 diatur oleh APB2 *timer clock*. *Timer-timer* yang dimiliki oleh STM32Fxx dapat dijadikan sebagai *input capture*, *output compare*, PWM, *One-pulse mode output,* DAC *trigger* dan membangkitkan sinyal DAC*,* maupun untuk pengunakan biasa (*general purpose*). Perhatikan diagram Gambar 32.



Gambar 31. APB *timer*



Gambar 32. Blok diagram *timer*

6.2. Timer Interrupt

Interrupt yang dapat digunakan oleh *timer* meliputi:

1) Update: *counter* overflow/underflow, *counter initialization* (oleh *software* atau *internal/external* trigger)
2) Trigger event (counter start, stop, initialization atau count by internal/external trigger)
3) Input capture
4) Output capture

Register yang digunakan untuk *time base* antara lain:

1) Counter Register (TIMx_CNT)
2) Prescaler Register (TIMx_PSC):
3) Auto-Reload Register (TIMx_ARR)

Rumus yang digunakan untuk menghitung waktu *overflow* adalah sebagai berikut:

$$T_{ovf} = \frac{(htim2.Init.Prescaler + 1) \times (htim2.Init.Period + 1)}{f_{timer\_clock}}$$

Contoh 1:

Jika *clock timer* = 10 MHz, *htim2.Init.Prescaler* = 9999 dan *htim2.Init.Period*=999, berapa waktu yang diperlukan oleh TIM2 untuk *overflow*?

Jawab:

$$T_{ovf} = \frac{(9999 + 1) \times (999 + 1)}{10000000} = 1\ sec$$

Contoh 2:

Jika frekuensi *clock timer* = 1 MHz, berapa nilai *htim2.Init.Prescaler* dan *htim2.Init.Period* jika diinginkan TIM2 *overflow* setiap 100 ms (0,1 sec)?

Jawab:

$$T_{ovf} = \frac{(htim2.Init.Prescaler + 1) \times (htim2.Init.Period + 1)}{1000000}$$
$$= 0{,}1\ sec$$

maka *htim2.Init.Prescaler* dan *htim2.Init.Period* dapat diisi berapa saja asalkan memenuhi rumus di atas dan tidak melebihi batas maksimalnya, yakni 65535 (2^{16}-1) untuk *htim2.Init.Prescaler* dan 4294967295 (2^{32}-1) untuk *htim2.Init.Period*. Misalnya:

htim2.Init.Prescaler = 99 dan *htim2.Init.Period* = 999, atau *htim2.Init.Prescaler* = 499 dan *htim2.Init.Period* = 199, atau jawaban lainnya.

6.3. Praktikum Bab 6

Alat dan Bahan

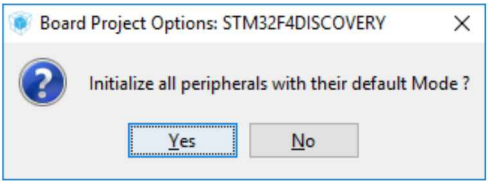
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Praktikum 6-01

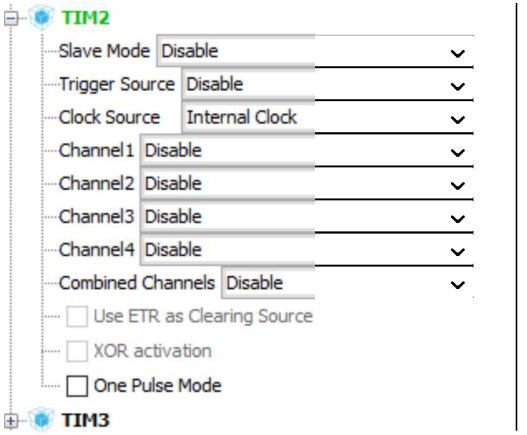
Pada praktikum ini, kita akan mendemonstrasikan *timer* dengan menggunakan LED biru (PD15). Tujuan program adalah membuat LED berkedip dengan periode 1 detik.

Ikuti langkah-langkah berikut:

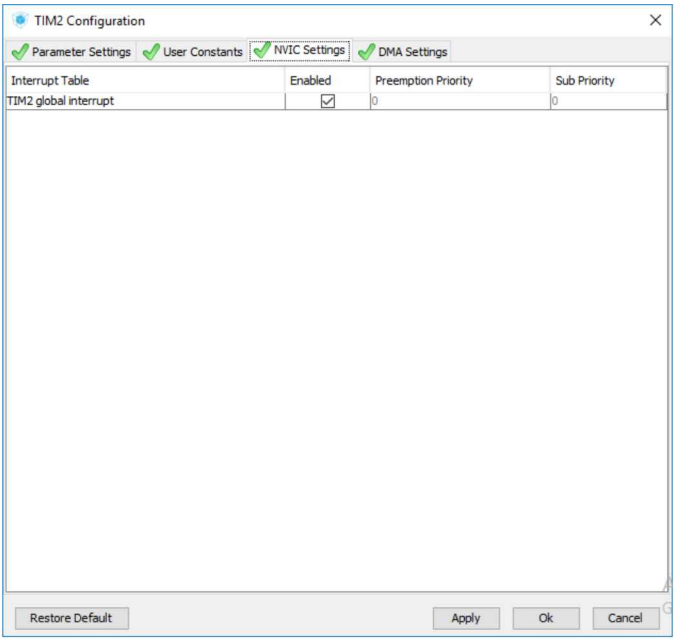
- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* Board Selector. Pada Part Number Search, pilih STM32F4DISCOVERY. Klik ganda STM32F4DISCOVERY.
- Klik No ketika muncul tampilan berikut:



- Set TIM2 sesuai gambar di bawah.



- Pada TIM2 Configuration, aktifkan TIM2 global interrupt. Kemudian klik OK.



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate Code. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c. Ubah kode pada fungsi static void MX_TIM2_Init(void) menjadi sebagai berikut. Perhatikan angka 6249 dan 999. Angka-angka inilah yang mempengaruhi waktu *overflow*.

```
/* TIM2 init function */
static void MX_TIM2_Init(void)
{
 TIM_ClockConfigTypeDef sClockSourceConfig;
 TIM_SlaveConfigTypeDef sSlaveConfig;
 TIM_MasterConfigTypeDef sMasterConfig;

 htim2.Instance = TIM2;
 htim2.Init.Prescaler = 6249;
 htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim2.Init.Period = 999;
 htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }

 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
 if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }

 sSlaveConfig.SlaveMode = TIM_SLAVEMODE_TRIGGER;
 sSlaveConfig.InputTrigger = TIM_TS_ITR2;
 if (HAL_TIM_SlaveConfigSynchronization(&htim2, &sSlaveConfig) != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }

 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
 if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }
}
```

- Pada main.c, tambahkan perintah untuk memulai *timer*. Perintah ini ditempatkan sebelum loop while(1).

```
HAL_TIM_Base_Start_IT(&htim2);
while (1)
{

}
```

- Buka stm32fxx_it.c dan temukan *interrupt handler*, yakni fungsi TIM2_IRQHandler(void). Tambahkan kode sehingga menjadi seperti berikut:

```
/**
* @brief This function handles TIM2 global interrupt.
*/
uint8_t sign = 0;
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */

  /* USER CODE END TIM2_IRQn 1 */
 if (sign == 0)
 {
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
  sign = 1;
 }
 else if (sign == 1)
 {
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
  sign = 0;
 }
}
```

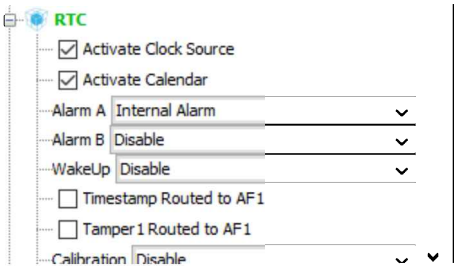
- Compile kode tersebut dengan cara klik Project | Build Target (atau dengan menekan icon  maupun menekan tombol F7 pada keyboard), kemudian tunggu hingga prosesnya selesai.
- Setelah proses compile selesai, set STLink *programmer* dengan cara klik Project | Options for Target 'Nama *File*' (atau dengan klik icon  maupun menekan tombol ALT+F7 pada keyboard).
- Sekarang, load program yang telah dibuat ke dalam mikrokontroler. Pastikan STM32F4DISCOVERY sudah terhubung dengan computer via USB. Klik Flash | Download (atau dengan menekan icon 

maupun menekan tombol F8 pada keyboard).
- Jika proses *load* program sukses, perhatikan LED biru yang berkelap-kelip dengan periode 1 detik.

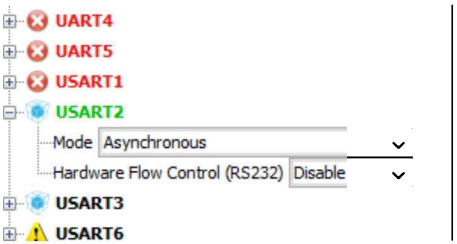
Praktikum 6-02

Pada praktikum ini, kita membuat *Log File* Menggunakan *Real Time Clock* (RTC). Lakukan langkah-langkah percobaan sebagai berikut:

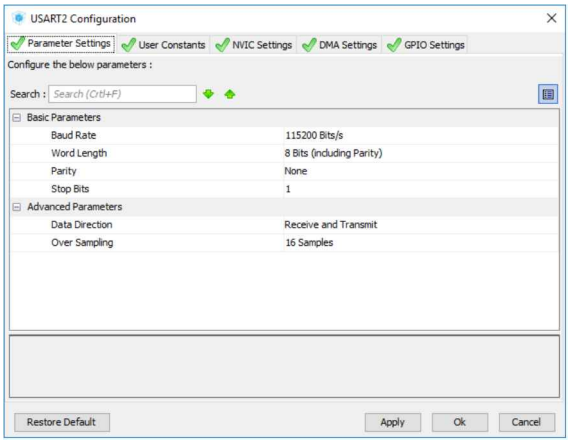
- Aktifkan RTC dengan checklist Activate *Clock* Source dan Activate Calendar. Kemudian pilih Internal Alarm pada Alarm A.



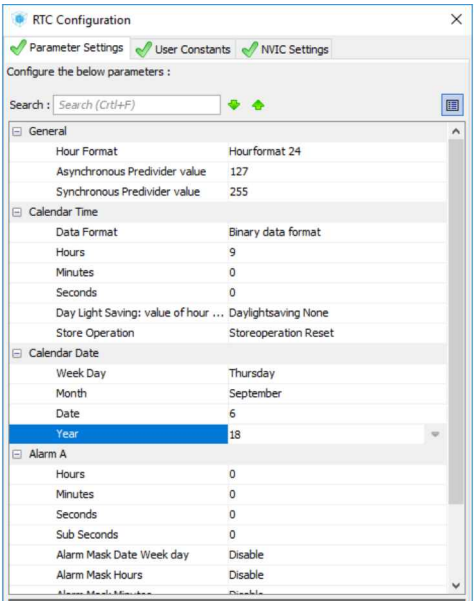
- Aktifkan USART2 dengan mode Asynchronous.



- Pada USART2 Configuration, konfigurasi UART seperti gambar di bawah.



- Buka RTC Configration. Isi seperti gambar di bawah. Pada Data Format, pilih Binary data format. Sesuaikan Date dan Time dengan waktu terkini. Untuk penulisan tahun, masukkan dua digit terakhir.



- *Generate code* dan buka Keil.
- Temukan deklarasi variabel sTime, sDate, dan sAlarm pada static void MX_RTC_Init(void). Kedua variabel ini merupakan variabel lokal yang hanya dapat diakses dalam fungsi tersebut, sehingga tidak dapat diakses oleh fungsi main(). Untuk itu, pindahkan deklarasi tersebut ke luar semua fungsi (misalnya pindahkan ke sebelum fungsi main()) sehingga menjadi variabel global.
- Pada praktikum ini, kita akan menggunakan fungsi printf(). Untuk itu, perlu dilakukan beberapa langkah. Masukkan kode berikut pada *header file*.

```
#include "main.h"
#include "stm32f4xx_hal.h"
#include "string.h"

#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif
```

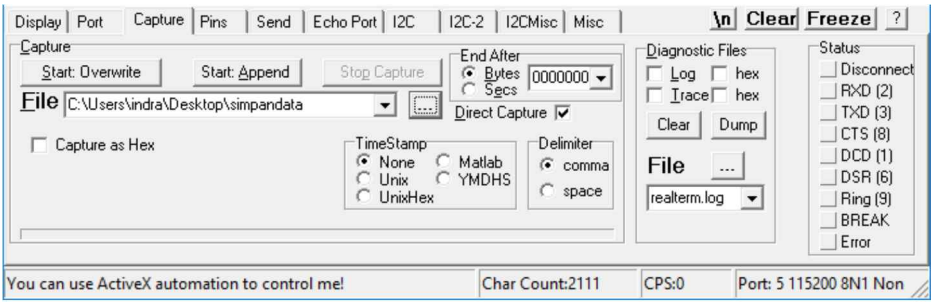
- Pada bagian Private function prototypes, tambahkan kode sehingga menjadi seperti berikut:

```
/* Private function prototypes -----------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_RTC_Init(void);

PUTCHAR_PROTOTYPE
{
 HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
 return ch;
}
```

- Masukkan kode berikut:

```
while (1)
{
 HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
 HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
 if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1)
 {
  printf("%d:%d:%d:  PRESSED\r\n", sTime.Hours, sTime.Minutes, sTime.Seconds);
 }
 else
 {
  printf("%d:%d:%d:  RELEASED\r\n", sTime.Hours, sTime.Minutes, sTime.Seconds);
 }

 HAL_Delay(1000);
}
```

- Pahami kode di atas. *Build* dan *load* program.
- Buka RealTerm. Set baudrate=115200, Display = Ansi. Sesuaikan nama *port*. Perhatikan apa yang tampil pada layar.
- Langkah selanjutnya adalah menyimpan data pada *file* txt. Pada RealTerm, buka *tab* Capture. Tentukan nama *file* tempat kita akan menyimpan *file*. Klik Start: Over*write* dan tunggu selama sekitar 20 detik.



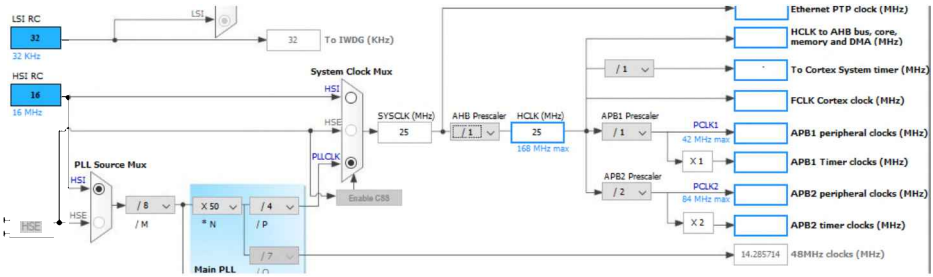
- Setelah itu, klik Stop Capture. Buka *file* txt tempat kita tadi menyimpan data dengan menggunakan aplikasi Notepad atau Wordpad. Perhatikan data apa yang terkandung di dalamnya?

6.4. Tugas Praktik Bab 6

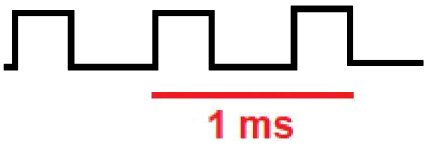
a) Buat periode *blinking* pada Praktikum 6-01 menjadi 500 ms dan 200 ms.
b) Tambahkan *file* pada Praktikum 6-02 sehingga pada *log file* terdapat informasi tanggal (DD/MM/YY).

6.5. Latihan Soal Bab 6

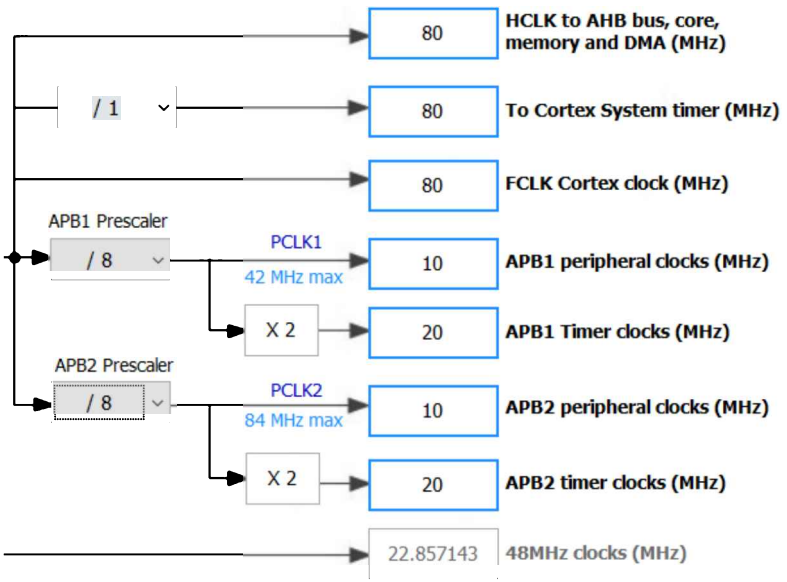
- Sebutkan *timer* pada STM32F407VG yang diatur oleh APB1!
- Berapa frekuensi APB2 *Timer Clock* (MHz) sesuai gambar di bawah?



- TIM2 memiliki *clock* 1 MHz. Jika *htim2.Init.Prescaler*=200 dan *tim2.Init.Period*=500, maka berapa waktu yang diperlukan oleh TIM2 untuk overflow?
- TIM2 memiliki *clock* 25 MHz, berapa nilai *htim2.Init.Prescaler* dan *htim2.Init.Period* jika diinginkan *Timer*2 overflow setiap 1 ms? (Jawaban boleh lebih dari satu variasi)
- Sebuah PWM memiliki duty cycle 40% (lihat gambar). Berapa frekuensi PWM tersebut?



- Sebuah PWM yang diatur oleh TIM2 memiliki *htim2.Init.Prescaler*=10 dan *htim2.Init.Period*=10000. Berapa % resolusi PWM tersebut?
- TIM2 memiliki *clock* 12,5 MHz, berapa nilai htim2.Init.Prescaler dan htim2.Init.Period jika diinginkan frekuensi PWM 1 kHz? (Jawaban boleh lebih dari satu)
- TIM2 memiliki $htim2.Init.Period$ = 10000. Berapa nilai CCR1 agar dihasilkan PWM dengan duty cycle 5%?

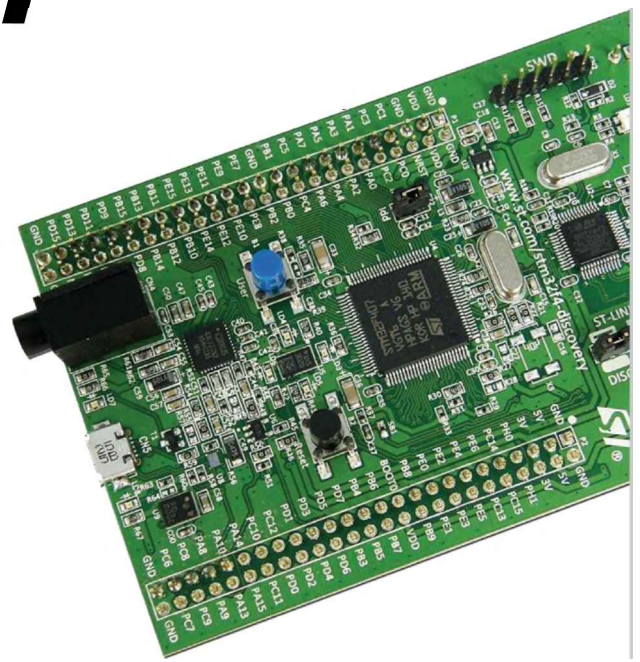


Gambar 33. Contoh konfigurasi *timer*

- Lihat Gambar 33. *htim2.Init.Prescaler*=1999 dan *htim2.Init.Period*=4999, berapa waktu yang diperlukan oleh TIM2 untuk overflow?
- Lihat Gambar 33. Berapa *htim2.Init.Prescaler* dan htim2.Init.Period jika diinginkan *Timer*2 overflow setiap 2 ms?

Sistem Embeded Berbasis ARM Cortex-M

7

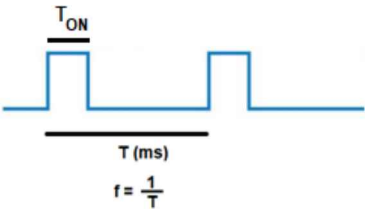


Bab 7 PWM

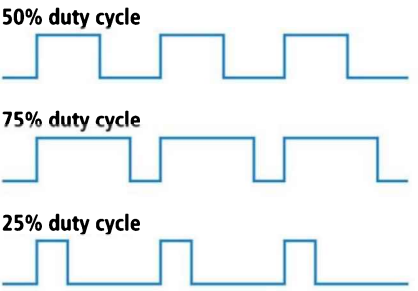
7.1. Prinsip Kerja PWM

Pulse Width Modulation (PWM) merupakan sebuah teknik modulasi yang mengkodekan (*encode*) nilai menjadi sinyal pulsa. Meskipun tujuan utamanya untuk pengkodean sebuah pesan ke dalam sinyal pulsa, namun teknik modulasi ini dapat juga digunakan untuk mengendalikan suplai daya ke peralatan elektrik/elektronik. Sinyal yang dihasilkan merupakan pulsa dengan periode dan *duty cycle* tertentu yang dapat diatur. T (Period) adalah $T_{overflow}$ pada bab sebelumnya, yakni waktu yang diperlukan *timer* untuk *overflow*. Sedangkan frekuensi dapat dihitung dengan menggunakan periode. *Duty cycle* (satuan %) didefinisikan sebagai perbandingan antara waktu pulsa ON (T_{ON}) dan periode. Nilai dari *duty cycle* berkisar antara 0% (*fully* OFF) dan 100% (*fully* ON). Perhatikan Gambar 34 dan Gambar 35.

$$Duty\ cycle\ (\%) = \frac{T_{ON}}{Periode}$$



Gambar 34. Periode PWM



Gambar 35. Periode dan duty cycle PWM[3]

STM32Fxx memiliki fitur *Pulse Width Modulation* (PWM) dengan memanfaatkan *timer* yang dimiliki. PWM memudahkan kita membuat sinyal dengan frekuensi dan *duty cycle* yang sudah ditentukan. Cara menentukan frekuensi adalah dengan memasukkan nilai ke *register* TIMx_ARR, sedangkan *duty cycle* diset dengan memasukkan nilai ke *register* TIMx_CCRx.

Resolusi pada PWM menunjukkan seberapa kecil kenaikan/penurunan nilai PWM yang memungkinkan. Resolusi PWM ini ditentukan oleh resolusi *timer* yang digunakan, misalnya *timer* 16-bit memiliki resolusi maksimal 100%÷65535 yakni 0,0015%, yang berarti duty cycle yang diberikan merupakan kelipatan 0,0015%. Walaupun demikian, resolusi yang digunakan tidak selalu sesuai dengan resolusi maksimal. Ini tergantung dari $htim2.Init.Period$ yang diberikan. Misalnya TIM2 (32-bit) dengan $htim2.Init.Period$ = 1000, maka akan menghasilkan resolusi 100%÷1000 yaitu 0,1%, artinya duty cycle yang diberikan adalah kelipatan 0,1% (yaitu 0%, 0,1%, 0,2%, ...99,9%, 100%). Pengaturan *duty cycle* inilah yang akan memengaruhi kecepatan motor, intensitas cahaya LED, nada pada *buzzer*, dan sebagainya.

Mengatur *duty cycle* berarti mengatur nilai CCRx (x=channel), sesuai rumus berikut:

$$Duty\ cycle(\%) = \frac{htim2.Instance{-}{>}\ CCR1}{htim2.Init.Period} \times 100\%$$

Misalnya: Jika $htim2.Instance{-}{>}\ CCR1$ = 10 dan $htim2.Init.Period$ = 1000, maka duty Cycle = 1%. Pada inisialisasi MCU, setting CCR ini dilakukan dengan kode **sConfigOC.Pulse**.

[3] Ini merupakan PWM non-inverting. Untuk PWM inverting, maka sinyal menjadi kebalikannya.

7.2. Praktikum Bab 7

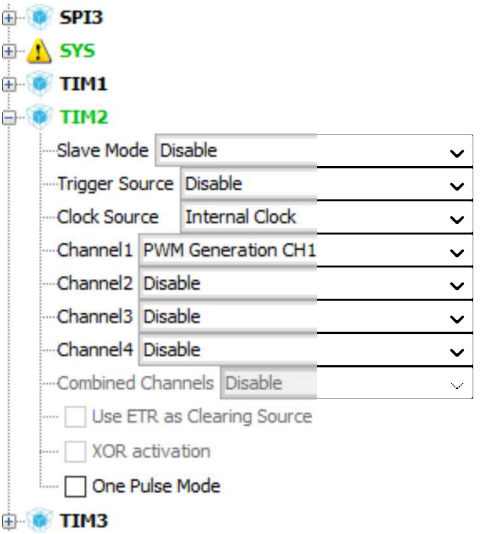
Alat dan Bahan

- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

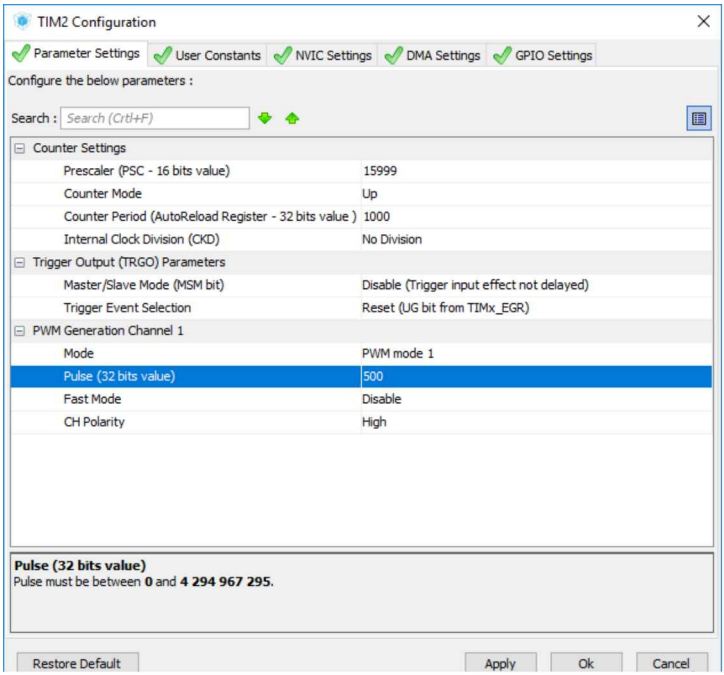
Pada praktikum ini, kita akan mendemonstrasikan PWM dengan menggunakan LED pada PD13. Tujuan program adalah membuat LED menyala dengan intensitas yang dapat diatur menggunakan PWM. Kali ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY

Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Gunakan TIM2 *channel* 1 untuk PWM:



- Konfigurasi TIM2 sebagai berikut:



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate Code. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c. Mulai PWM dengan cara masukkan kode berikut sebelum *super loop* while(1) pada fungsi main(). Hal ini dilakukan untuk memulai *timer*.

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

- Hubungkan (menggunakan kabel *jumper*) kaki PA0 dan PD13 pada STM32F4DISCOVERY.
- *Build* dan *load* program.
- Perhatikan nyala pada LED orange.
- Coba ubah-ubah angka pada sConfigOC.Pulse dengan angka 0 s.d 999, misalnya: 100, 200, dst. Apa pengaruh angka-angka ini dengan nyala LED?
- Hitung berapa frekuensi PWM pada praktikum ini?

7.3. Tugas Praktik Bab 7

Tugas 701 ini menggunakan pengetahuan mengenai UART *interrupt* pada pelajaran sebelumnya. Buat sebuah program untuk membuat sinyal PWM dengan frekuensi 2kHz. Duty cycle dapat diatur dengan memasukkan angka menggunakan UART sesuai dengan tabel berikut:

Angka yang dimasukkan	Duty cycle
0	0%
1	10%
2	20%
3	30%
4	40%
5	50%
6	60%
7	70%
8	80%
9	90%
10	100%

Pengujian dilakukan dengan 2 cara:

1) Amati sinyal dengan menggunakan osiloskop; dan
2) gunakan MCU dan rangkaian tambahan untuk mengontrol kecepatan motor DC, buzzer, atau LED[4].

Petunjuk:

Untuk memasukkan nilai **sConfigOC.Pulse** (duty cycle), maka dapat menggunakan kode:

```
htim2.Instance->CCR1 = a;
```

a merupakan bilangan bulat $0 \leq a \leq htim2.Period$

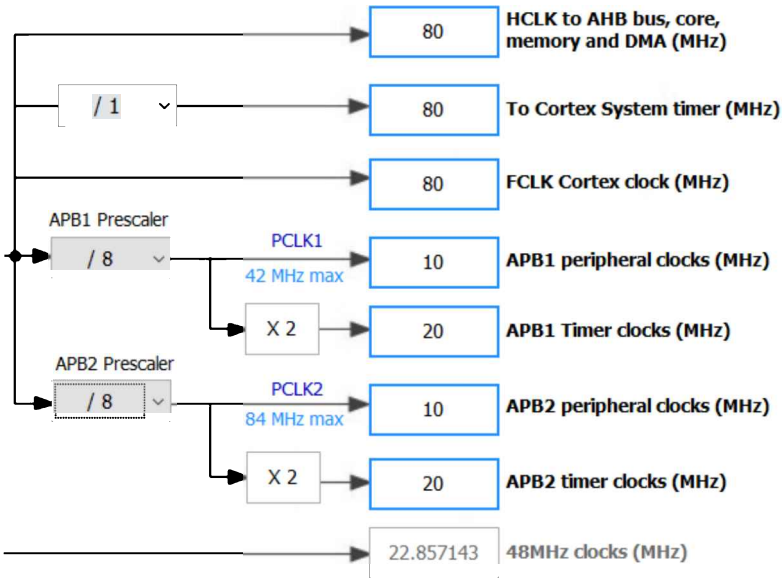
Contoh:

```
htim2.Instance->CCR1 = 500;
```

[4] *Pilihan Motor DC atau LED sesuai dengan ketersediaan bahan praktikum. Ikuti rangkaian yang diberikan dosen*

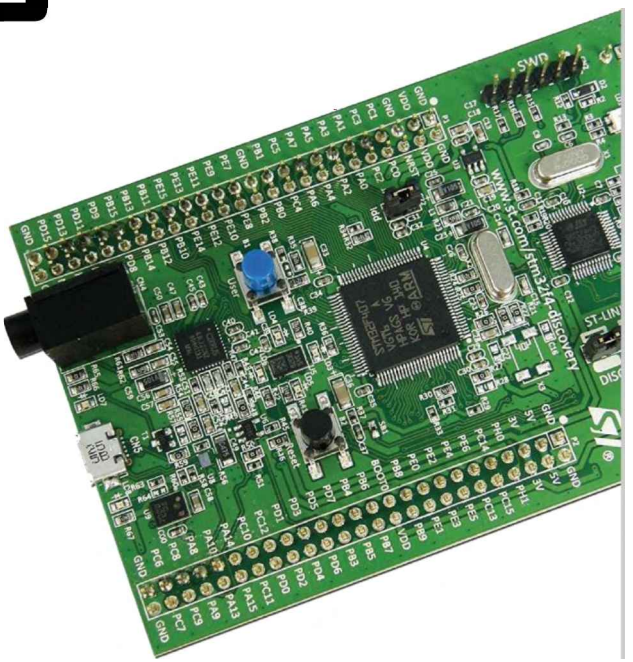
7.4. Latihan Soal Bab 7

- Apa yang dimaksud dengan *duty cycle*?
- Jika PWM digunakan untuk mengatur kecepatan motor DC, apa efek dari nilai frekuensi dan *duty cycle* pada putaran motor tersebut?
- Berapa nilai *htim2.Init.Prescaler*, $htim2.Instance{->}CCR1$ dan $htim2.Init.Period$ agar diperoleh PWM dengan frekuensi 500 Hz dan *duty cycle* 30%?



Sistem Embeded Berbasis ARM Cortex-M

8



Bab 8 ADC

Analog to Digital Converter (ADC) adalah salah satu fitur pada mikrokontroler untuk mengubah sinyal analog menjadi data *digital*. Fitur ini menjadi jembatan antara dunia analog (transduser) dan dunia *digital* untuk pemrosesan data lebih lanjut. Contoh aplikasi ADC dapat dijumpai pada mikrofon, modul strain gauge, modul thermocouple, multimeter *digital*, osiloskop *digital*, timbangan, dan sebagainya.

8.1. Proses Konversi ADC dan Perhitungannya

Secara garis besar, proses konversi analog ke *digital* ini melalui 2 tahap:

1) Sampling dan holding
2) Quantization dan encoding

Resolusi dari ADC ditentukan oleh berapa bit kemampuan ADC tersebut, sedangkan kecepatan sampling ADC ditentukan oleh sampling rate dengan satuan *Sampling per Second* (SPS).

ADC biasanya memiliki *Error*, sebagai contoh:

1) *Aliasing*: terjadi karena *sampling rate* lebih rendah dari frekuensi sinyal yang dikonversi. *Nyquist Rule* menyatakan bahwa sampling rate setidaknya 2 kali lipat dari frekuensi yang dikonversi untuk mencegah aliasing ini.
2) *Quantization Error*: diakibatkan resolusi ADC yang terbatas

Perhitungan ADC (D_{IN}) dari sebuah sinyal analog (V_{IN}) tergantung dari resolusi (N) dan tegangan referensi (V_{ref}) dengan rumus sebagai berikut:

$$D_{IN} = \frac{V_{IN}}{V_{ref}} \times (2^N - 1)$$

Contoh:

Diketahui:

Resolusi ADC 3 bit

Tegangan referensi (V_{ref}) = 5 Volt

Tegangan *input* (V_{IN}) = 1,2 Volt

Ditanyakan: Berapa nilai ADC nya (D_{IN})?

Jawab:

$D_{IN} = \frac{2,7V}{5V} \times (2^3 - 1)$ = 3,78 → dibulatkan menjadi 3

Pembulatan ke bawah (*round down*) dilakukan karena D_{IN} merupakan bilangan bulat (*integer*).

8.2. ADC pada STM32F4

STM32F4 memiliki ADC 16 *channel* eksternal yang terhubung dengan pin I/O. Berikut adalah tabel yang menunjukkan ADC dan channel yang terhubung ke pin:

Tabel 15. ADC dan *channel* yang terhubung ke pin

Channel	ADC1	ADC2	ADC3
APB	2	2	2
ADC Channel 0	PA0	PA0	PA0
ADC Channel 1	PA1	PA1	PA1
ADC Channel 2	PA2	PA2	PA2
ADC Channel 3	PA3	PA3	PA3
ADC Channel 4	PA4	PA4	PF6
ADC Channel 5	PA5	PA5	PF7
ADC Channel 6	PA6	PA6	PF8
ADC Channel 7	PA7	PA7	PF9
ADC Channel 8	PB0	PB0	PF10
ADC Channel 9	PB1	PB1	PF3

ADC Channel 10	PC0	PC0	PC0
ADC Channel 11	PC1	PC1	PC1
ADC Channel 12	PC2	PC2	PC2
ADC Channel 13	PC3	PC3	PC3
ADC Channel 14	PC4	PC4	PF4
ADC Channel 15	PC5	PC5	PF5

Selain itu, terdapat pula 3 *channel* internal yang terbagi menjadi:

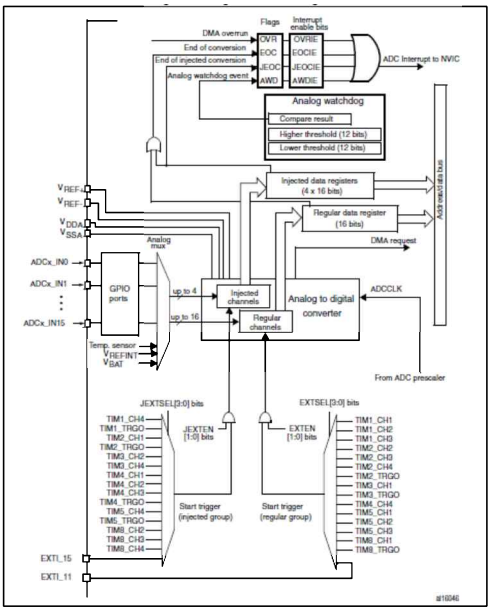
- V_{bat} : tegangan pada pin baterai untuk RTC (*Real Time Clock*)
- Temp Sensor: Tidak dapat digunakan untuk mengukur suhu, hanya untuk mengukur perbedaan suhu karena dapat gagal sampai 45 ° C
- V_{ref}: Tegangan referensi untuk ADC

Secara diringkas, pin-pin tersebut dijelaskan dalam Tabel 16.

Tabel 16. Pin-pin ADC

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, 1.8 V ≤V_{REF+} ≤V_{DDA}
V_{DDA}	Input, analog supply	Analog power supply equal to V_{DD} and 2.4 V ≤V_{DDA} ≤V_{DD} (3.6 V) for full speed 1.8 V ≤V_{DDA} ≤V_{DD} (3.6 V) for reduced speed
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, V_{REF-} = V_{SSA}
V_{SSA}	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog input signals	16 analog input channels

ADC pada STM32F4 memiliki resolusi maksimal 12-bit. Pengguna dapat memilih mode penggunaannya apakah konversi dilakukan secara *single* atau *continuous*. Gambar 36 menggambarkan diagram blok dari ADC.



Gambar 36. Diagram blok dari ADC

Fitur-fitur ADC pada STM32F4 adalah sebagai berikut:

- 16 *channel*: [15:0]
- Resolusi yang dapat dipilih: 12-bit, 10-bit, 8-bit or 6-bit
- Mode konversi:
- *Single mode*: ADC melakukan satu kali konversi
- *Continuous mode*: ADC otomatis melakukan konversi yang baru setelah konversi sebelumnya selesai
- *Scan*: ADC melakukan scan pada satu grup *channel* analog
- *Discontionuous*: ADC melakukan n konversi dalam waktu yang singkat (n≤8)
- ADC interrupt
- Tegangan suplai ADC: 2.4 V - 3.6 V pada *full speed* dan 1.8 V pada *low speed*
- Tegangan *input* (V_{IN}): $V_{REF-} \leq V_{IN} \leq V_{REF+}$

ADC terbagi atas sirkuit analog dan *Interface digital*. *Clock* untuk keduanya merupakan *clock* APB2 yang dapat dibagi dengan *prescaler* (fPCLK2/2, /4, /6, atau /8). Waktu yang digunakan untuk konversi (T_{conv}) merupakan jumlah dari waktu *sampling* ($Sampling\_Time$)

dan siklus minimum ($Minimum\_Cycle$), sesuai dengan rumus berikut:

$$T_{conv} = Sampling\_Time + Minimum\_Cycle$$

$Minimum\_Cycle$ tergantung resolusi yang digunakan sesuai angka pada Tabel 17. Sebagai contoh ADC 12-bit, jika ADCCLK = 30 MHz dan waktu sampling = 3 *cycle*, maka T_{conv} adalah 3 + 12 = 15 *cycle*, atau 0,5 µs.

Tabel 17. Minimum *cycle*

Resolusi	*Minimum_Cycle*
12-bit	12
10-bit	10
6-bit	8
8-bit	6

8.3. Praktikum Bab 8

Alat dan Bahan

- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

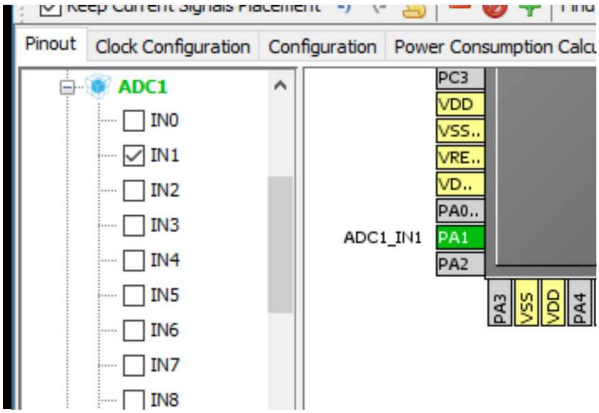
Praktikum 8-01

Pada praktikum ini, kita akan mendemonstrasikan ADC dengan menggunakan Single Mode. Tujuan program adalah mengubah data analog (disimulasikan dengan menggunakan potensiometer) menjadi *digital*. Kali ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY.

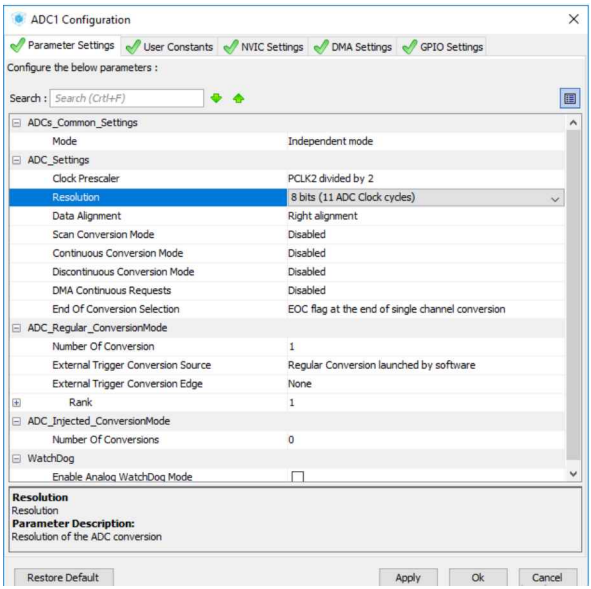
Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.

- Gunakan ADC1 – IN1 (PA1):



- Konfigurasi ADC1 sebagai berikut:

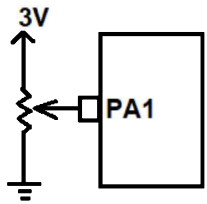


- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate *Code*. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open *Project*, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.

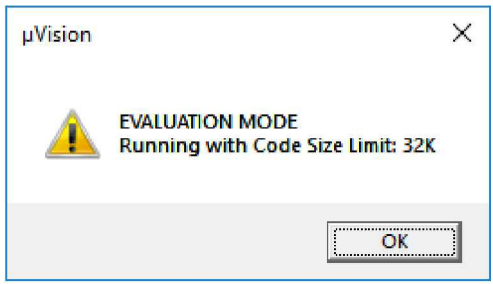
- Deklarasikan variabel baru bernama nilaiAdc dan buat kode sebagai berikut:

```
uint8_t nilaiAdc;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();

  while (1)
  {
        HAL_ADC_Start(&hadc1);
        if (HAL_ADC_PollForConversion(&hadc1, 5)==HAL_OK)
        {
         nilaiAdc = HAL_ADC_GetValue(&hadc1);
        }
        HAL_ADC_Stop(&hadc1);
        HAL_Delay(50);
  }
}
```

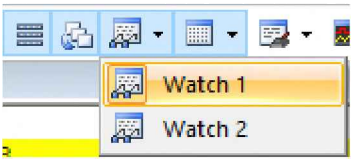
- *Build* dan *load* program.
- Buat rangkaian potensiometer sebagai berikut:



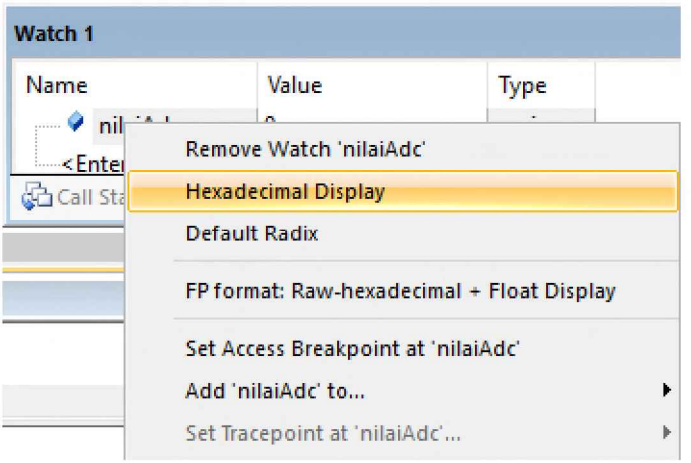
- Klik tombol Start/Stop Debug Session , kemudian klik OK jika ada *message box* sebagai berikut:



- Cari tombol Watch Windows dan Klik Watch1.



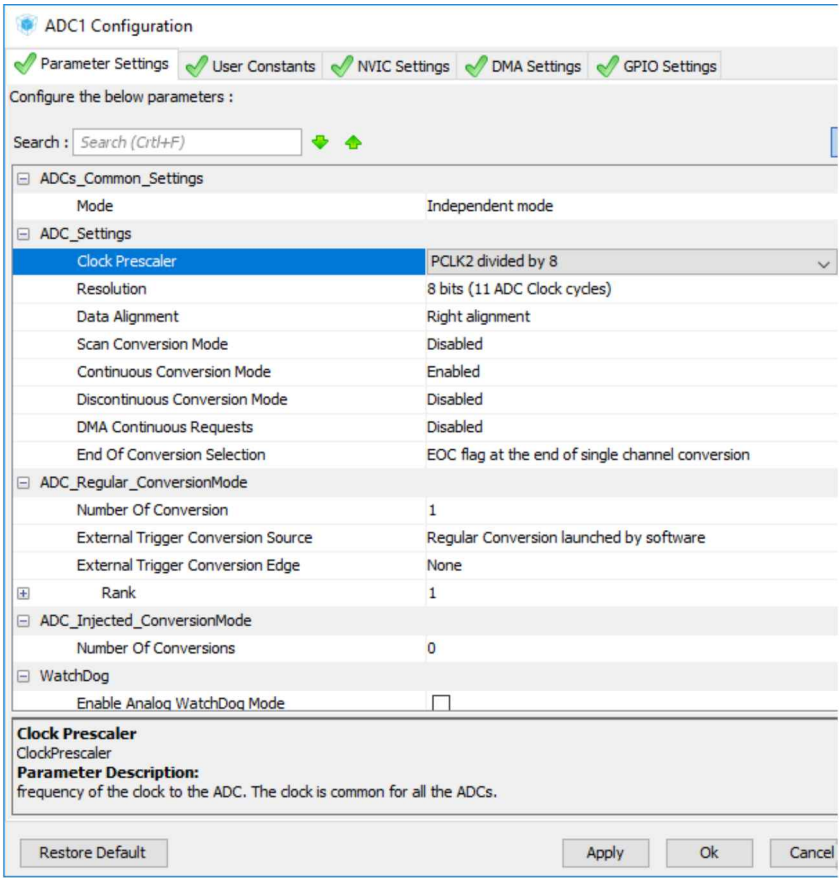
- Ketik nilaiADC pada Watch1. Kemudian klik kanan pada nilaiAdc dan *uncheck* Hexadecimal Display.



- Klik tombol Run  (atau tekan F5) dan perhatikan perubahan pada nilaiAdc.

Praktikum 8-02

Pada praktikum ini, kita akan menggunakan **ADC Continuous Mode**. Lakukan langkah seperti pada praktikum 801. Namun ubah konfigurasi ADC menjadi sebagai berikut:



- Tambahkan kode berikut pada main.c:

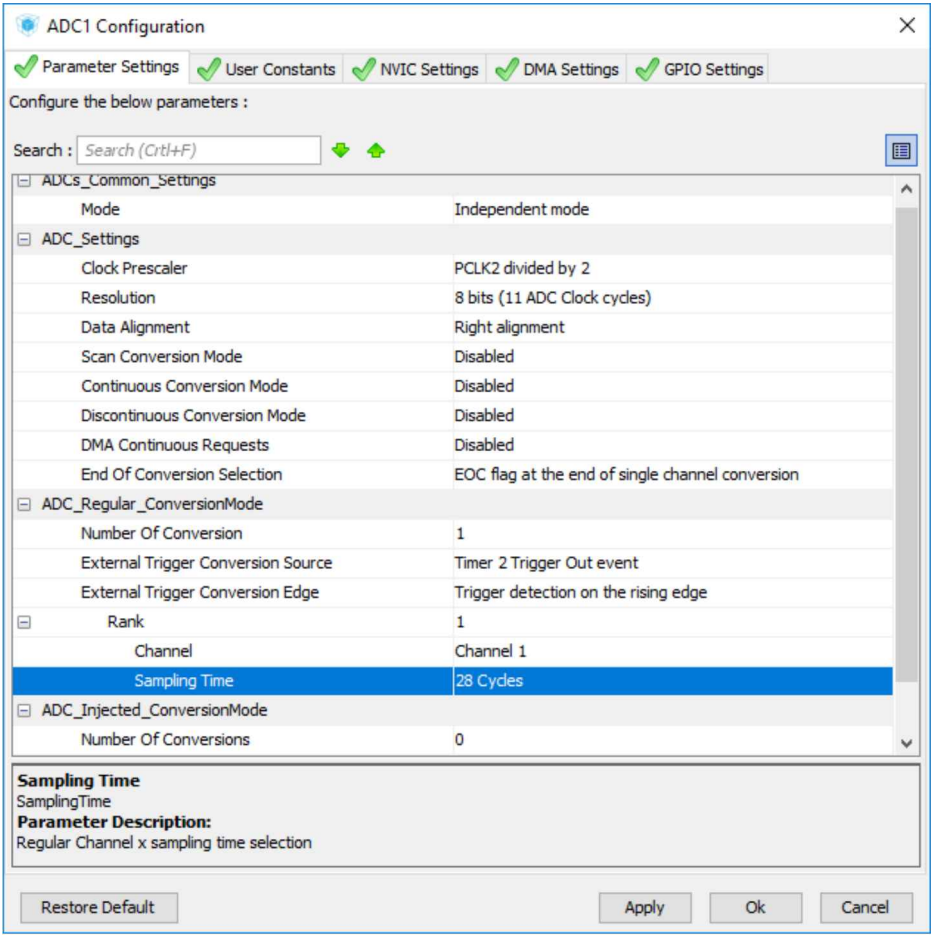
```
uint8_t nilaiAdc;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  HAL_ADC_Start(&hadc1);

  while (1)
  {
        nilaiAdc = HAL_ADC_GetValue(&hadc1);
  }
}
```

- *Build* dan *load* program, kemudian lakukan *debug*ging.

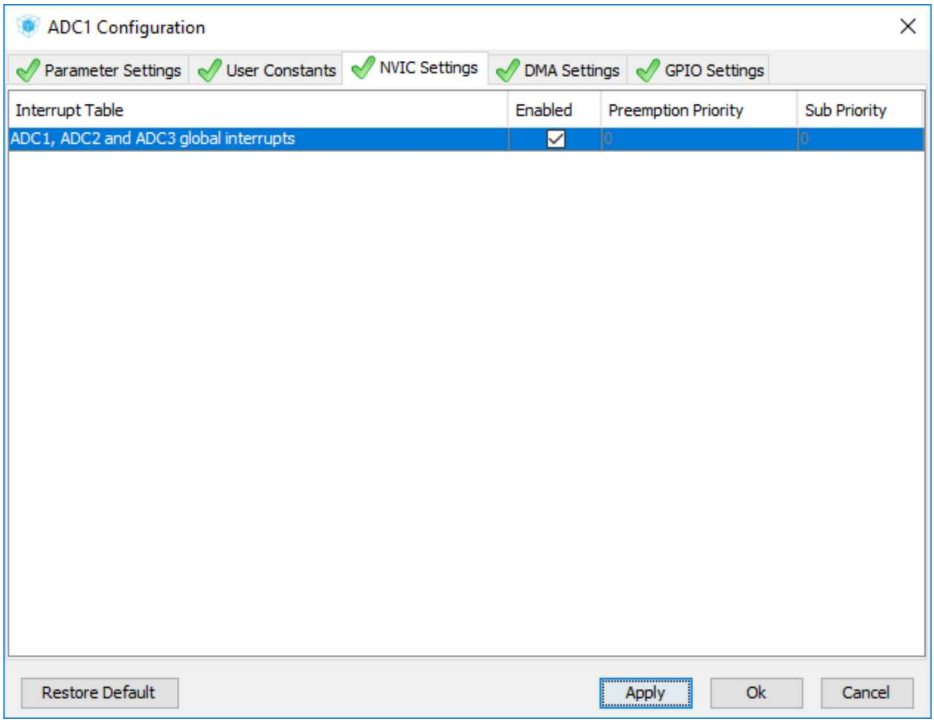
Praktikum 8-03

Pada praktikum ini, konversi ADC dilakukan secara periodik dengan periode yang dapat diatur menggunakan *timer*. Lakukan langkah seperti pada praktikum 801. Namun ubah konfigurasi ADC menjadi sebagai berikut:

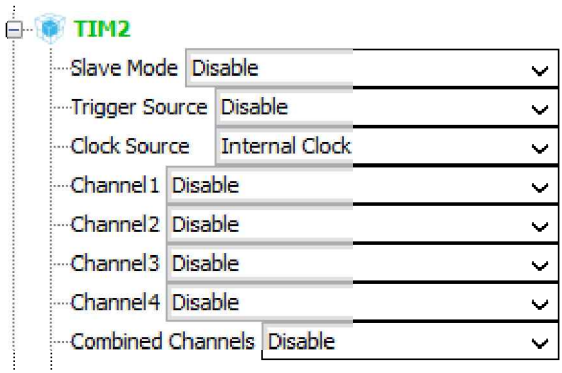
- Konfigurasi ADC sebagai berikut:



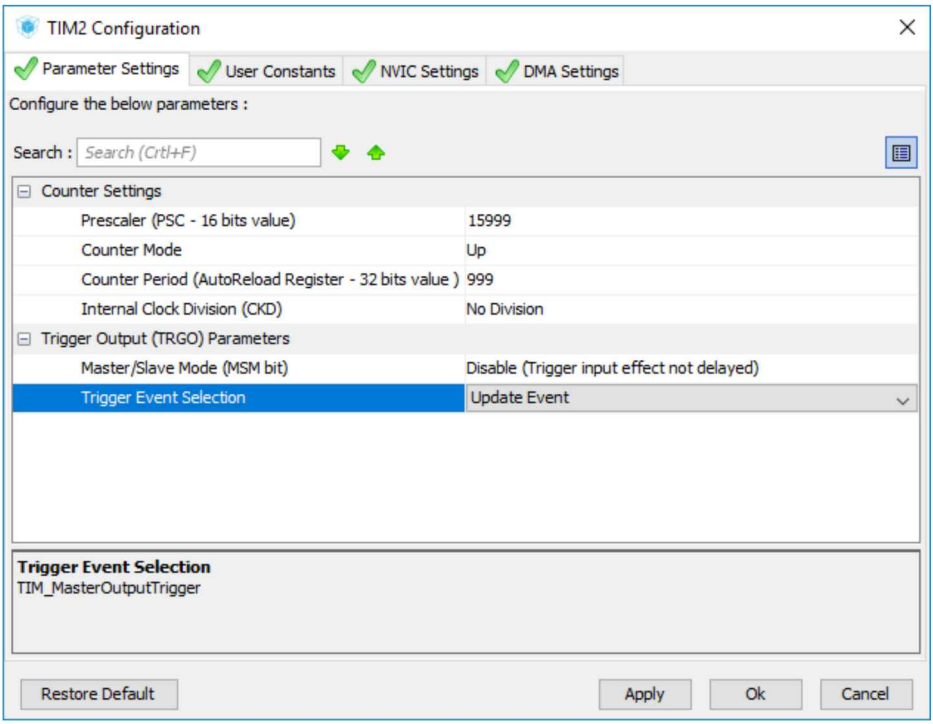
- Aktifkan ADC *Interrupt*:



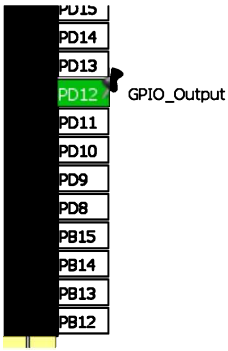
- Tambahkan *timer*. Gunakan TIM2:



- Set TIM2 sebagai berikut:



- Set PD12 sebagai *output* (*push pull*):



- Tambahkan kode berikut pada main.c:

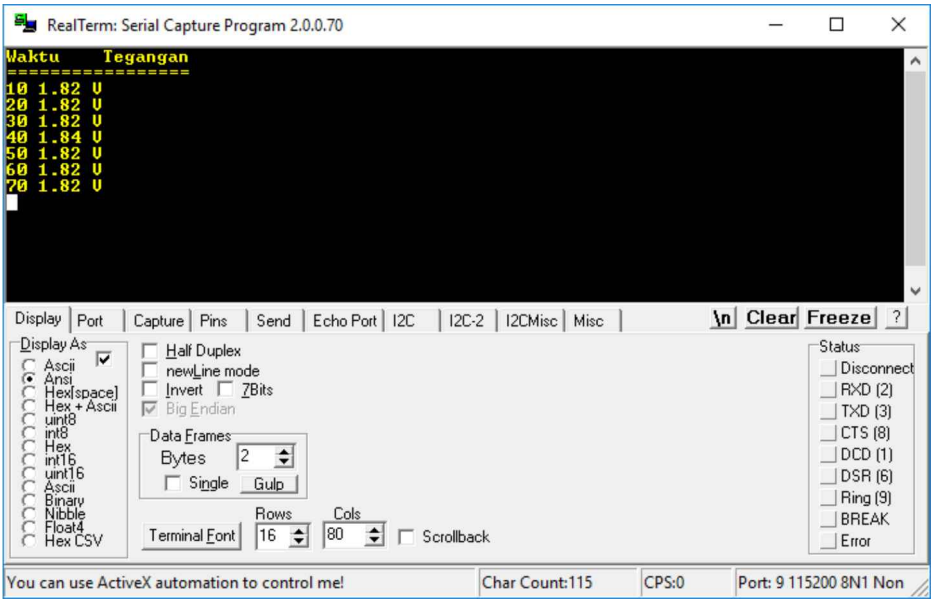
```
uint8_t nilaiAdc;
uint32_t counter;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();

  HAL_TIM_Base_Start(&htim2);
  HAL_ADC_Start_IT(&hadc1);
  while (1)
  {
  }
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
        UNUSED(hadc);
        nilaiAdc = HAL_ADC_GetValue(&hadc1);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
}
```

- *Build* dan *load* program, kemudian lakukan *debug*ging.
- Pahami kodenya. Apa perbedaan dengan Praktikum 8-01 dan 8-02?

8.4. Tugas Praktik Bab 8

Pada Tugas ini, mahasiswa membuat data logger yang dilengkapi dengan informasi waktu. Lakukan langkah-langkah berikut:

- Set ADC sebagaimana pada Praktikum 8-03 (menggunakan *timer*). Gunakan periode 10 ms.
- Aktifkan UART2 dan tambahkan kode sehingga fungsi printf() dapat dijalankan.
- Pada main.c, konversi nilai ADC yang masuk ke dalam satuan volt dengan resolusi 2 angka di belakang koma. (NB: angka harus dalam float untuk pembagian desimal)
- Tambahkan informasi waktu dengan menggunakan fungsi HAL_GetTick(). Ini merupakan fungsi untuk mengetahui nilai Tick dalam millisecond.
- Tampilkan waktu dan tegangan pada RealTerm seperti gambar berikut:

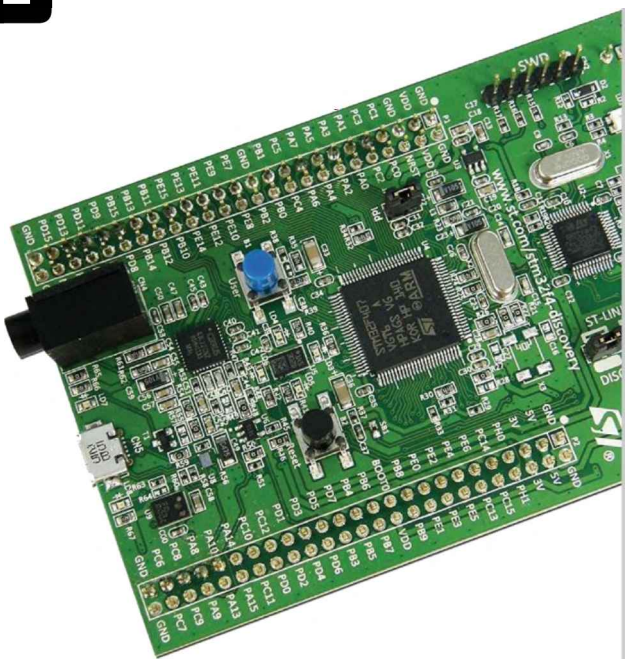


8.5. Latihan Soal Bab 8

- Sebuah ADC memiliki resolusi 8 bit dan tegangan referensi 3V. Berapa nilai ADC (D_{IN}) jika tegangan *input*nya 2,5V?
- Sebuah MCU membaca hasil ADC (10-bit) sebesar 102. Berapa Volt aktual tegangan yang sedang diukur jika MCU tersebut menggunakan tegangan referensi internal 1,1V?
- Seorang teknisi ingin mengukur sinyal sinusoidal dengan frekuensi sekitar 1GHz. ADC dengan kemampuan sampling minimal berapa SPS yang harus dia beli?
- ADC diketahui memiliki kemampuan *sampling* 5 MSPS. Berapa detik jarak antar *sampling* yang satu terhadap sampling berikutnya?

Sistem Embeded Berbasis ARM Cortex-M

9



Bab 9 DAC

9.1. Prinsip Kerja DAC

Digital to Analog Converter (DAC) adalah alat yang mengonversikan kode *digital* menjadi analog, baik tegangan maupun arus. Contoh aplikasi DAC antara lain kendali motor *digital*, printer, MP3 p*Layer*, dan lain sebagainya.

Ketika memilih sebuah DAC, ada beberapa spesifikasi yang perlu diperhatikan, antara lain:

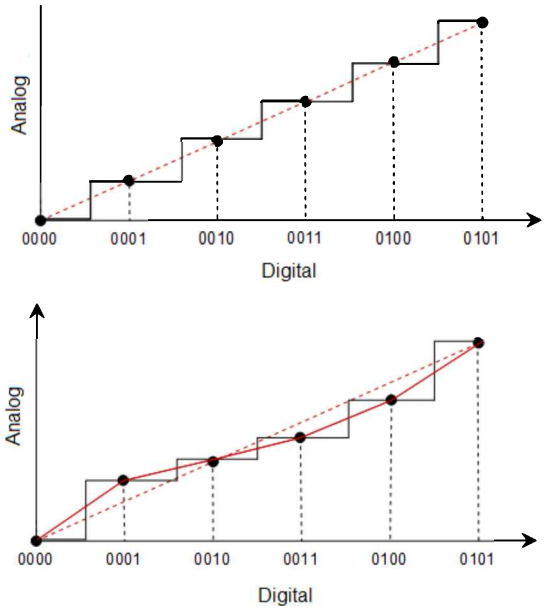
1) Resolusi
2) Kecepatan
3) Linearitas
4) Settling time
5) Tegangan referensi
6) Error

Resolusi pada DAC dapat diartikan sebagai kenaikan nilai analog untuk setiap kenaian 1 *Least Significat Bit* (LSB). Semakin besar bit yang digunakan (N), maka semakin tinggi resolusi yang dihasilkan terhadap tegangan referensi (V_{ref}), sesuai rumus berikut:

$$Resolusi = V_{LSB} = \frac{V_{ref}}{2^N - 1}$$

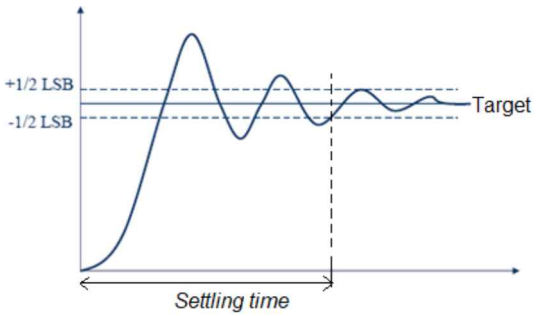
Kecepatan pada DAC menggambarkan berapa waktu yang diperlukan untuk melakukan satu kali konversi dari *input digital* ke *output* analog. Kecepatan ini tergantung pada kecepatan *clock* dari sinyal *input* dan *Settling time* dari DAC. Jika *input digital* berubah dengan cepat, maka konversi DAC harus juga cepat untuk mengimbanginya.

Linearitas pada DAC menggambarkan perbedaan antara sinyal analog yang diharapkan dengan yang dihasilkan secara aktual. Idealnya grafik yang dihasilkan merupakan grafik linear sempurna, namun terkadang dihasilkan *Error* yang memengaruhi Linearitas ini. Perhatikan Gambar 37.



Gambar 37. Linearitas pada DAC. Atas: linear, bawah: tidak *line*ar

Settling time didefinisikan sebagai waktu yang diperlukan DAC untuk menghasilkan sinyal *output* analog yang stabil pada ±½ LSB setelah nilai *digital*nya berubah. Semakin baik kualitas DAC, maka semakin cepat *Settling time* ini. Idealnya, perubahan sinyal *output* analog terjadi tepat setelah perubahan sinyal *digital*nya. Perhatikan Gambar 38.



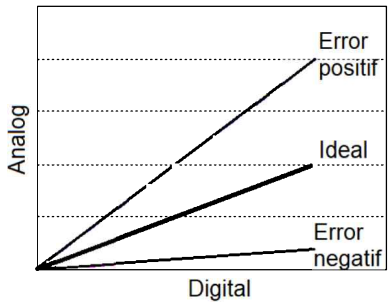
Gambar 38. *Settling* time

Sebagaimana ADC, tegangan referensi menentukan besarnya sinyal analog yang dihasilkan dari *input digital*. Tegangan referensi ada yang tetap (*fixed*) maupun berasal dari eksternal.

Ada beberapa jenis *Error* yang mungkin dialami oleh DAC, di antaranya:

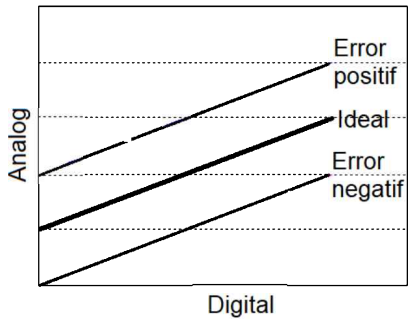
1) Gain
2) Offset
3) Full Scale
4) Resolution
5) Non-*Line*arity
6) Non-Monotonic
7) *Settling* Time *and* Overshoot

Gain Error merupakan kesalahan pada DAC di mana kemiringan (*slope*) yang dihasilkan berbeda dengan yang diharapkan. Perhatikan Gambar 39.



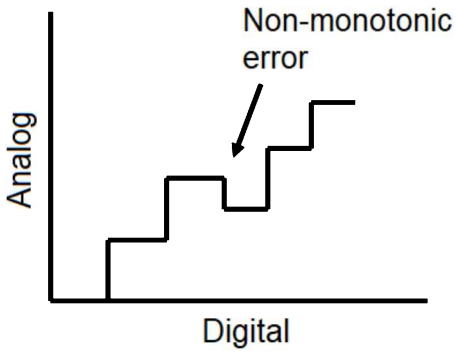
Gambar 39. Gain *Error*

Offset Error terjadi ketika muncul offset yang konstan antara *output* ideal dengan *output* aktual. Perhatikan Gambar 40.



Gambar 40. Offset *Error*

Jika *gain* dan *offset Error* terjadi secara bersamaan, maka *Error* tersebut dinamakan *full scale Error*. *Resolution Error* terjadi karena keterbatasan yang dimiliki oleh DAC, sebagaimana juga dialami oleh ADC. Semakin besar jumlah bit nya, maka semakin tinggi resolusinya dan semakin kecil resolution *Error* nya. *Non-line*arity *Error* merupakan *Error* yang terjadi karena masalah Linearitas pada DAC, sebagaimana dijelaskan sebelumnya pada Gambar 37. *Non-monotic Error* terjadi ketika kenaikan pada nilai *digital* malah menurunkan nilai *output* analog (yang seharusnya ikut naik). Perhatikan Gambar 41.



Gambar 41. *Non-monotonic Error*

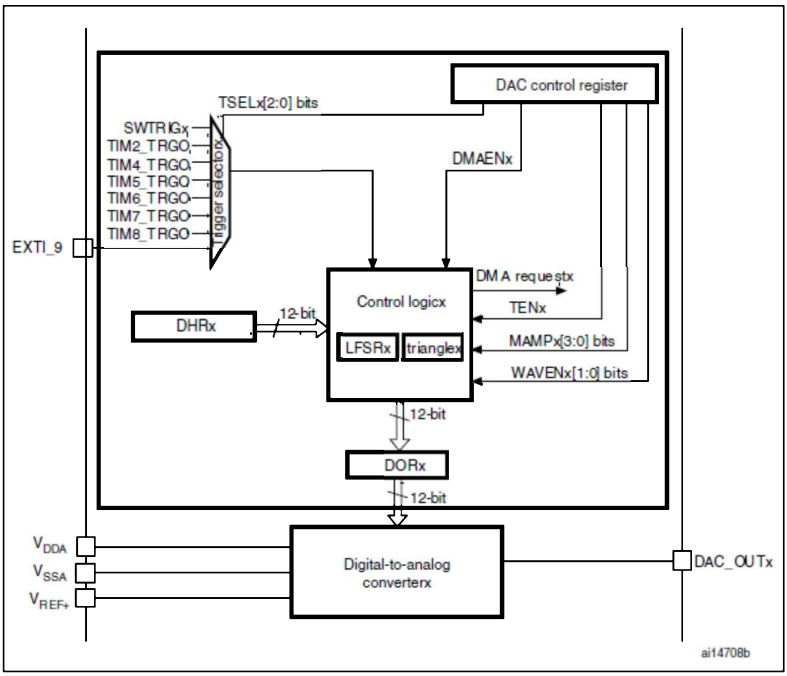
Settling time Error terjadi ketika DAC memerlukan waktu untuk menghasilkan sinyal *output* analog yang stabil pada ±½ LSB setelah nilai *digital*nya berubah sebagaimana dijelaskan sebelumnya pada Gambar 38. Idealnya *Settling* time bernilai 0. *Overshoot Error* terjadi saat *output* analog melebihi nilai *output* ideal (Gambar 38).

9.2. DAC pada STM32F4

STM32F4 memiliki 2 *channel* DAC dengan resolusi masing-masing 12-bit. Berikut adalah fitur dari DAC yang dimiliki STM32F4:

- 2 channel DAC, 1 *output* per channel, konversi terpisah maupun bersamaan
- 12 bit DAC
- Kemampuan sinkronisasi
- Membangkitkan sinyal noise (sinyal pseudo random)
- Membangkitkan sinyal segitiga
- Dapat menggunakan DMA
- *Trigger* eksternal untuk melakukan konversi
- Menyediakan *input* untuk tegangan referensi eksternal VREF+

Diagram blok dari DAC dijelaskan dalam Gambar 42. Sedangkan pin-pin yang berkaitan dijelaskan pada Tabel 18. *Register-register* yang berhubungan langsung dengan DAC dijelaskan dalam Tabel 19.



Gambar 42. Diagram blok DAC

Tabel 18. Pin-pin DAC

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the DAC, 1.8 V ≤ V_{REF+} ≤ V_{DDA}
V_{DDA}	Input, analog supply	Analog power supply
V_{SSA}	Input, analog supply ground	Ground for analog power supply
DAC_OUTx	Analog output signal	DAC channelx analog output

Tabel 19. *Register-register* DAC

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	DAC_CR	Reserved		DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE 2[2:0]		TSEL2[2:0]			TEN2	BOFF2	EN2	Reserved		DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE 1[2:0]		TSEL1[2 :0]			TEN1	BOFF1	EN1
0x04	DAC_ SWTRIGR	Reserved																														SWTRIG2	SWTRIG1
0x08	DAC_ DHR12R1	Reserved																				DACC1DHR[11:0]											
0x0C	DAC_ DHR12L1	Reserved																DACC1DHR[11:0]														Reserved	
0x10	DAC_ DHR8R1	Reserved																								DACC1DHR[7:0]							
0x14	DAC_ DHR12R2	Reserved																				DACC2DHR[11:0]											
0x18	DAC_ DHR12L2	Reserved																DACC2DHR[11:0]														Reserved	
0x1C	DAC_ DHR8R2	Reserved																								DACC2DHR[7:0]							
0x20	DAC_ DHR12RD	Reserved				DACC2DHR[11:0]												Reserved				DACC1DHR[11:0]											
0x24	DAC_ DHR12LD	DACC2DHR[11:0]												Reserved				DACC1DHR[11:0]														Reserved	
0x28	DAC_ DHR8RD	Reserved																DACC2DHR[7:0]								DACC1DHR[7:0]							
0x2C	DAC_ DOR1	Reserved																				DACC1DOR[11:0]											
0x30	DAC_ DOR2	Reserved																				DACC2DOR[11:0]											
0x34	DAC_SR	Reserved		DMAUDR2		Reserved														DMAUDR1		Reserved											

Input digital (DOR) dikonversi menjadi sinyal analog (DAC_{Output}) dengan menggunakan rumus sebagai berikut:

$$DAC_{Output} = V_{ref} \times \frac{DOR}{4095}$$

9.3. Praktikum Bab 9

Alat dan Bahan

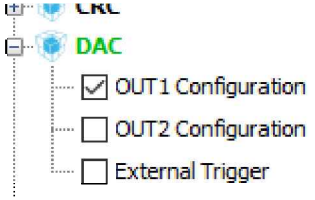
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Pada praktikum ini, kita akan mendemonstrasikan DAC 8 bit. Tujuan program adalah mengubah data *digital* menjadi analog. Kali

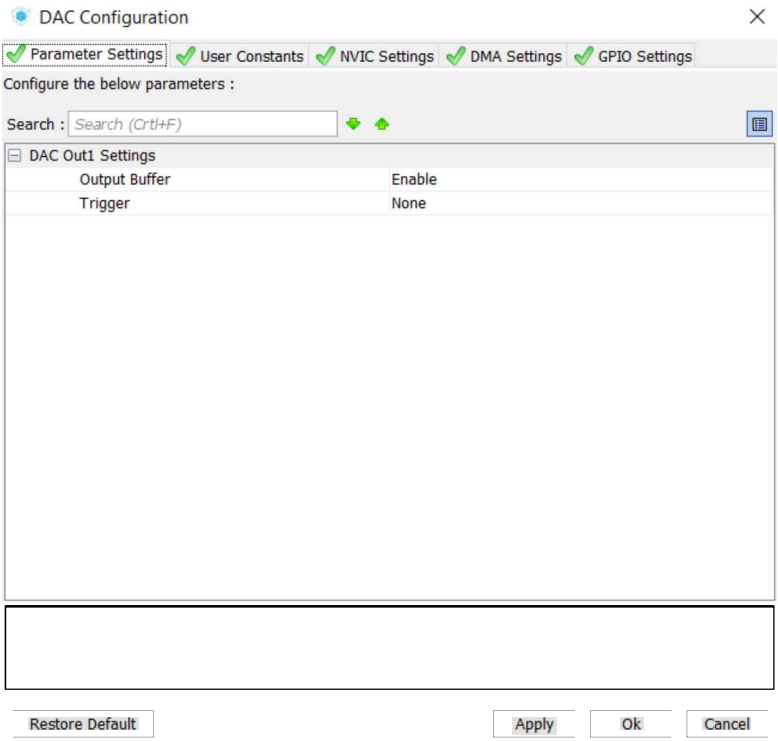
ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY.

Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Gunakan DAC1 – IN1 (PA4):



- Konfigurasi DAC1 sebagai berikut:



- Klik Project | Settings.

- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate *Code* dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Deklarasikan variabel baru bernama dacVolt dan dacByte dan modifikasi kode pada main.c sebagai berikut:

```
uint8_t dacByte
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DAC_Init();

  dacByte = 123;

  HAL_DAC_Start(&hdac DAC1_CHANNEL_1);
  HAL_DAC_SetValue(&hdac DAC1_CHANNEL_1 DAC_ALIGN_8B_R  dacByte);

  while (1)
  {
  }
```

- *Build* dan *load* program.
- Pahami kodenya. Ukur dengan menggunakan multimeter. Berapakah tegangan yang dihasilkan? Berapa yang seharusnya (menurut perhitungan)? Jika tidak sama, apa saja kemungkinan penyebabnya?

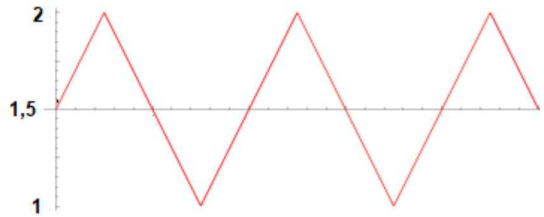
9.4. Tugas Praktik Bab 9

Tugas 9-01

Pada tugas ini, mahasiswa diminta menghasilkan tegangan DC 2 Volt. Berapa nilai DOR agar secara teoritis tegangan yang dihasilkan adalah 2 Volt? Program MCU dengan menggunakan DOR tersebut. Jika tegangan yang dihasilkan tidak 2 V, *adjust* nilai DOR agar tegangan *output* sedekat mungkin dengan 2 Volt.

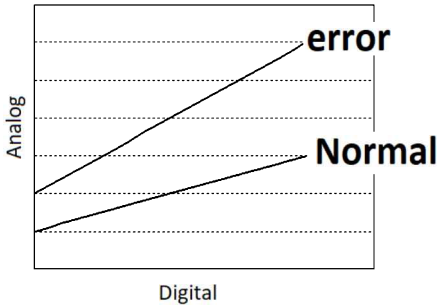
Tugas 9-02

Buat gelombang segitiga dengan amplitudo 0,5 volt dan tegangan bias 1,5 V. Tampilkan di osiloskop. Besarnya frekuensi sinyal tidak diatur, diserahkan ke mahasiswa.



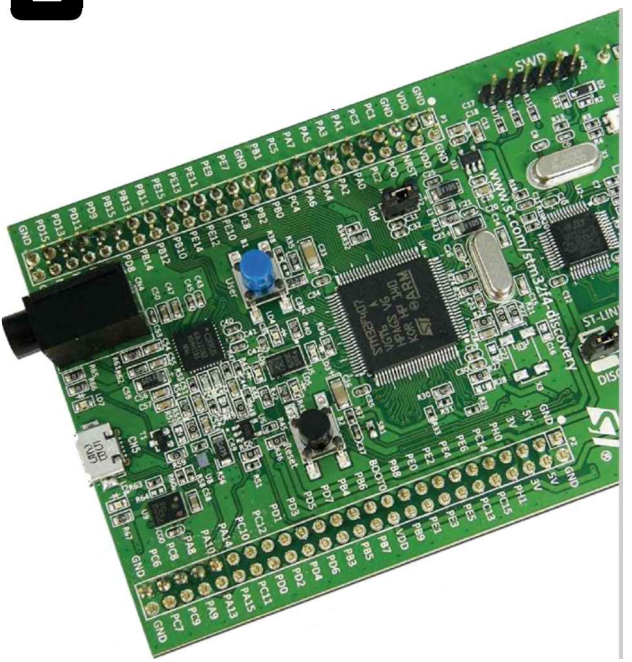
9.5. Latihan Soal Bab 9

- Sebuah DAC 8-bit memiliki nilai 64. Jika Vref nya 3,3V, berapa volt kira-kira tegangan yang dihasilkan?
- Sebutkan beberapa hal yang menjadi pertimbagan kita dalam memilih DAC!
- Apa saja jenis kesalahan pada DAC dan mana yang paling sering ditemukan menurut pengalaman Anda?
- *Error* pada DAC di bawah ini merupakan … *Error*.



Sistem Embeded Berbasis ARM Cortex-M

10



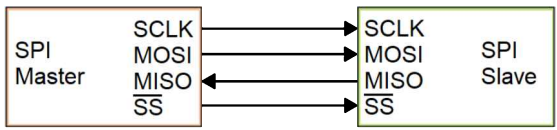
Bab 10 SPI

10.1. Prinsip Kerja SPI

Serial Peripheral Interface (SPI) merupakan komunikasi serial sinkron yang ditujukan untuk aplikasi jarak dekat, misalnya antar IC di dalam PCB yang sama. SPI memungkinkan kita melakukan komunikasi *half duplex* atau *full duplex* antar IC. Dalam SPI, satu alat diset sebagai *master*, sedangkan yang lainnya diset sebagai *slave* (topologi *master-slave*). SPI ini sendiri dikembangkan oleh Motorola pada tahun 1980-an dan menjadi standar sampai saat ini. SPI sangat banyak digunakan, misalnya untuk *Interface* modul sensor, ADC, DAC, Liquid *Crystal* Display (LCD), SD *card*, dan sebagainya.

SPI berkomunikasi secara *full-duplex*, yakni dua arah secara bersamaan. Topologi yang digunakan adalah *master-slave* dengan *master* tunggal dan beberapa *slave*. *Master* berfungsi mengatur komunikasi, baik pengiriman maupun penerimaan data. Dalam satu waktu, *master* hanya berkomunikasi dengan satu *slave* saja. Jika memiliki *slave* lebih dari satu, maka *master* memilih *slave* dengan cara mengeset pin *slave-select* ($\overline{SS}$), yang bersifat *active-low*. SPI menggunakan 4 sinyal yang digambarkan pada Gambar 43:

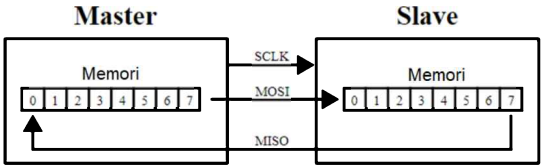
1) SCLK (Serial *Clock*): diatur oleh *master*
2) MOSI (*Master Out Slave In*): data dari *master* yang dikirim ke *slave*
3) MISO (*Master In Slave Out*): data dari *slave* yang dikirim ke *master*
4) $\overline{SS}$ (*Slave Select*): digunakan *master* untuk memilih *slave* yang akan berkomunikasi



Gambar 43. Komunikasi SPI

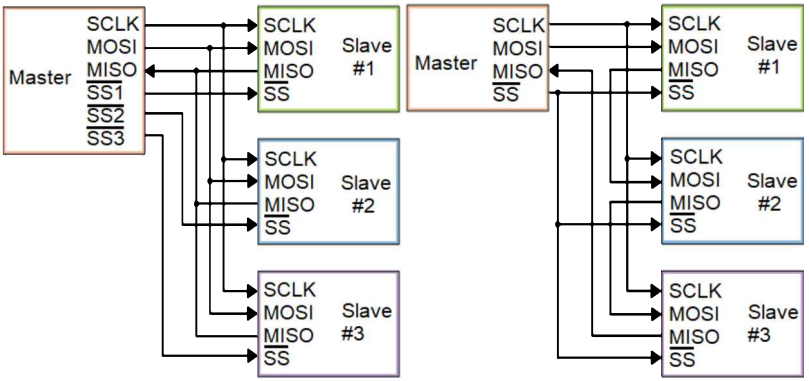
Untuk memulai komunikasi, *master* mengaktifkan SCLK dengan frekuensi yang disanggupi oleh *slave*. Kemudian, *master* memilih *slave* yang akan diajak berkomunikasi dengan mengaktifkan $\overline{SS}$ (memberikan logika 0). Komunikasi *full-duplex* pun terjadi. *Master*

mengirimkan data ke *slave* melalui MOSI dan pada saat yang bersamaan *slave* mengirimkan data ke *master* melalui MISO. Dengan kata lain, terjadi pertukaran data antara *master* dan *slave*. Contoh yang ditampilkan pada Gambar 44 merupakan pertukaran data sebanyak 8 bit. Pada gambar tersebut, pertukaran data selesai setelah 8 *clock* SCK.



Gambar 44. Pertukaran data sebanyak 8 bit menggunakan SPI

Pada SPI, satu *slave* terkoneksi dengan satu $\overline{SS}$, sehingga akan banyak kabel yang diperlukan jika *slave* yang terkoneksi cukup banyak. Untuk itu, topologi Daisy *chain* (Gambar 45) sering digunakan.



Gambar 45. Topologi *master slave*. Kiri: *independent slave*, kanan: Daisy *chain*.

Keuntungan menggunakan SPI di antaranya:

- Kecepatan transfer lebih cepat dibandingkan UART
- *Hardware* sangat sederhana, seperti shift *register*
- Mendukung multi *slave*
- Tidak terbatas data 8-bit, dapat lebih dari 8
- *Slave* menggunakan *clock* dari *master*, sehingga tidak memerlukan osilator yang presisi

- *Slave* tidak memerlukan alamat unik, sebagaimana I2C
- Hanya perlu menggunakakan 4 pin pada IC

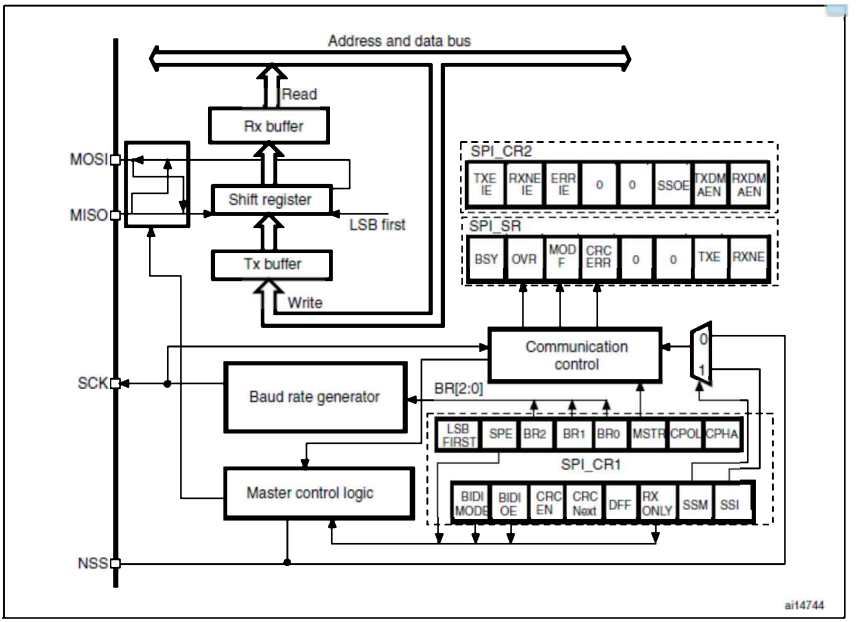
Sedangkan kelemahan menggunakan SPI antara lain:

- Memerlukan lebih banyak pin dibandingkan SPI
- Biasanya hanya memperbolehkan satu *master* saja
- Digunakan untuk jarak yang pendek, dalam satu PCB
- *Master* harus mengendalikan semua alur komunikasi, sehingga tidak memungkinkan komunikasi antar *slave*.
- Pada konfigurasi *independent slave* pin SS yang diperlukan sejumlah *slave* yang terhubung, sehingga memerlukan banyak pin jika jumlah *slave*nya banyak.

10.2. SPI pada STM32F4

Diagram blok SPI pada STM32F4 dapat dilihat pada Gambar 46. SPI pada STM32F4 memiliki fitur-fitur sebagai berikut:

- 3 buah SPI dengan kecepatan 42 Mbits/s (SPI1) dan 21 Mbits/s (SPI2 dan SPI3)
- Format frame: 8-bit atau 16-bit
- Dapat beroperasi sebagai *master* atau *slave*
- Polaritas dan fase *clock* (CPOL dan CPHA) yang dapat diprogram
- Urutan pengiriman data dapat diprogram, apakah MSB atau LSB yang lebih dulu dikirim
- *Flag* untuk menandakan bahwa pengiriman/penerimaan data telah selesai dilengkapi dengan pilihan untuk *trigger interrupt*. SPI memiliki beberapa *interrupt* request yang dijelaskan pada Tabel 20.
- *Flag* untuk menandakan status SPI *busy*

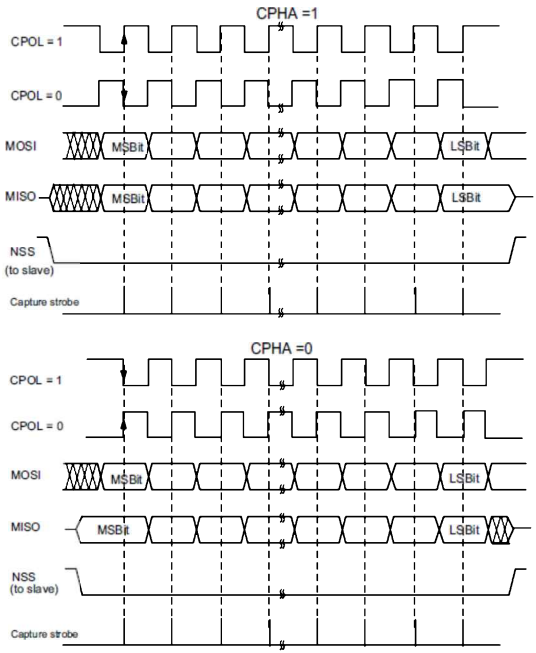


Gambar 46. Diagram blok SPI

Tabel 20. *Interrupt request* pada SPI

Interrupt event	Event flag	Enable Control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
CRC error flag	CRCERR	
TI frame format error	FRE	ERRIE

Pengguna dapat memilih 4 pilihan *timing* dengan menggunakan bit CPOL dan CPHA pada *register* SPI_CR1. *Timing diagram* dapat dilihat pada Gambar 47.



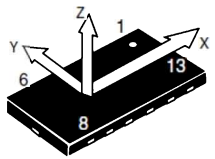
Gambar 47. *Timing diagram* pada SPI

10.3. Praktikum Bab 10

Alat dan Bahan

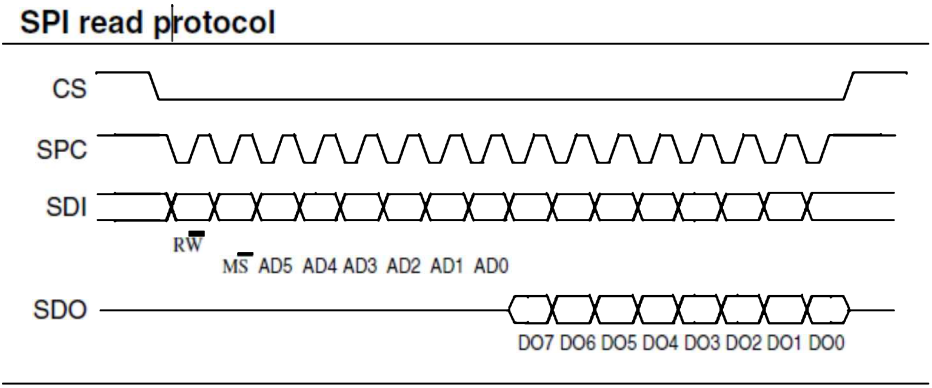
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Pada praktikum ini, kita akan mendemonstrasikan bagaimana membaca sensor akselerometer LIS3DSH (atau LIS302DL) menggunakan SPI (Gambar 48).

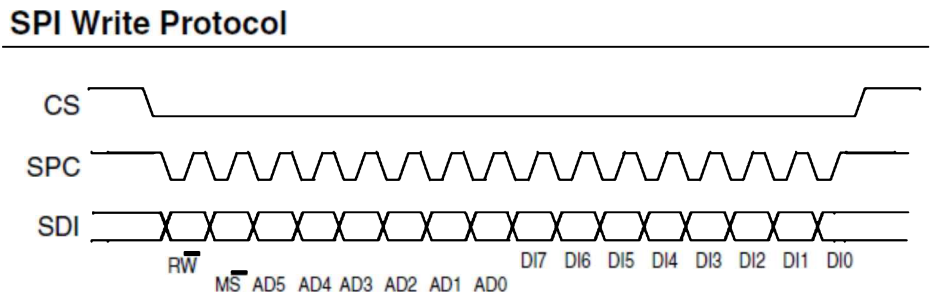


Gambar 48. LIS302DL

Protokol SPI saat membaca dan menulis digambarkan pada Gambar 49 dan Gambar 50, di mana $R\overline{W}$ = 1 (*read*) saat MCU membaca data dari akselerometer dan $R\overline{W}$ = 0 (*write*) saat MCU menulis data ke akselerometer.

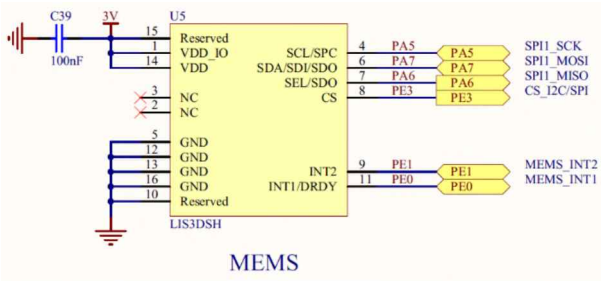


Gambar 49. SPI saat MCU membaca data dari LIS302DL

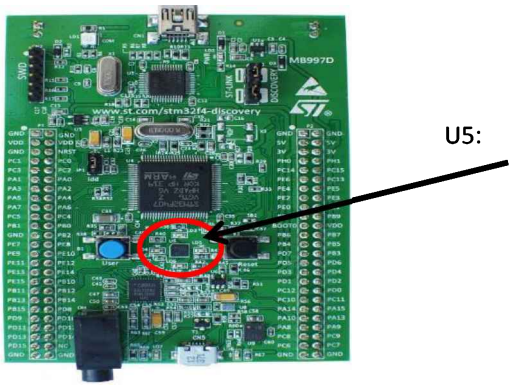


Gambar 50. SPI saat MCU menulis data ke LIS302DL

Koneksi akselerometer tersebut pada board STM32F4DISCOVERY dapat dilihat pada Gambar 51, sedangkan letaknya dapat dilihat pada Gambar 52.



Gambar 51. Koneksi akselerometer pada STM32F4DISCOVERY



Gambar 52. Letak MEMS pada STM32F4DISCOVERY

Register-register yang digunakan pada LIS30DL adalah seperti berikut (Tabel 21):

Tabel 21. *Register* pada LIS30DL

Name	Type	Register Address		Default	Comment
		Hex	Binary		
Reserved (Do not modify)		00-0E			Reserved
Who_Am_I	r	0F	00 1111	00111011	Dummy register
Reserved (Do not modify)		10-1F			Reserved
Ctrl_Reg1	rw	20	10 0000	00000111	
Ctrl_Reg2	rw	21	10 0001	00000000	
Ctrl_Reg3	rw	22	10 0010	00000000	
HP_filter_reset	r	23	10 0011	dummy	Dummy register
Reserved (Do not modify)		24-26			Reserved
Status_Reg	r	27	10 0111	00000000	
--	r	28	10 1000		Not Used
Out_X	r	29	10 1001	output	
--	r	2A	10 1010		Not Used
Out_Y	r	2B	10 1011	output	
--	r	2C	10 1100		Not Used
Out_Z	r	2D	10 1101	output	

Salah satu *register* yang digunakan untuk praktikum ini adalah CTRL_REG1 dan *register* data. Perhatikan Gambar 53 dan Gambar 54.

CTRL_REG1 (20h)

Control register #1.

DR	PD	FS	ST P	ST M	Zen	Yen	Xen

DR	Data rate selection. Default value: 0 (0: 100 Hz output data rate; 1: 400 Hz output data rate)
PD	Power Down Control. Default value: 0 (0: power down mode; 1: active mode)
FS	Full Scale selection. Default value: 0 (0: +/- 2g; 1: +/- 8g)
STP, STM	Self Test Enable. Default value: 00 (00: normal mode; 10: self test P; 01 self test M; 11: forbidden)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

Gambar 53. CTRL_REG1 pada LIS302DL

OUTX (29h)

X-axis output register.

XD7	XD6	XD5	XD4	XD3	XD2	XD1	XD0

OUTY (2Bh)

Y-axis output register.

YD7	YD6	YD5	YD4	YD3	YD2	YD1	YD0

OUTZ (2Dh)

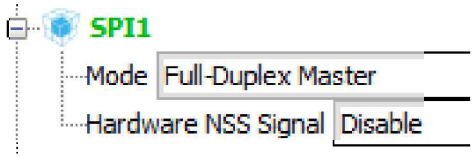
Z-axis output register.

ZD7	ZD6	ZD5	ZD4	ZD3	ZD2	ZD1	ZD0

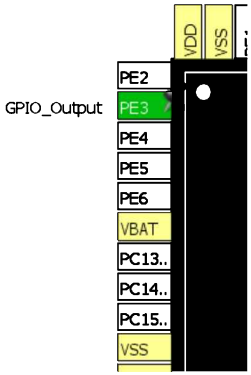
Gambar 54. *Register* data (OUTX, OUTY, dan OUTZ) pada LIS302DL

Setelah memahami cara menggunakan LIS302DL tersebut, ikuti langkah-langkah praktikum berikut:

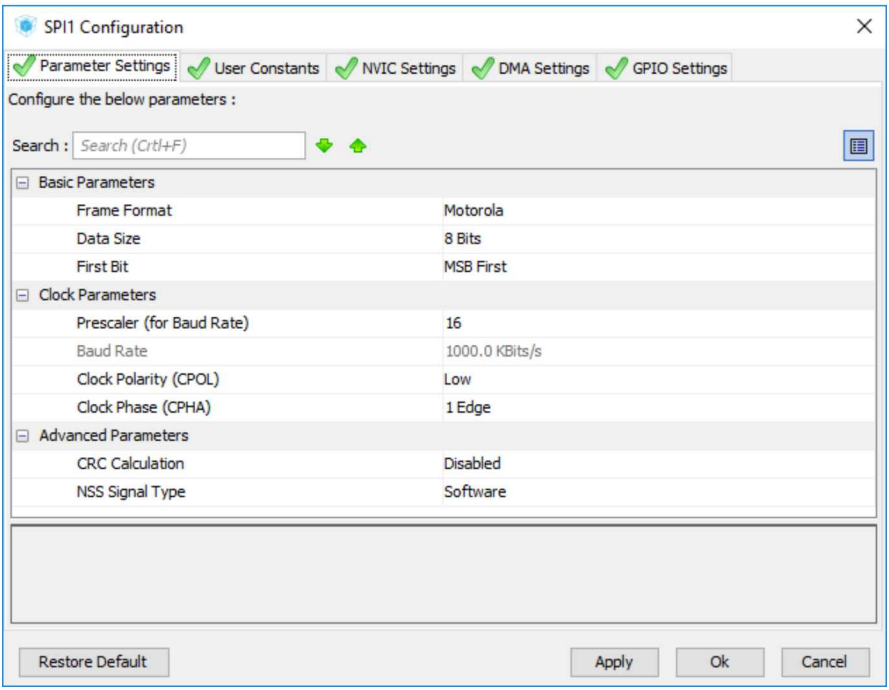
- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Gunakan SPI1:



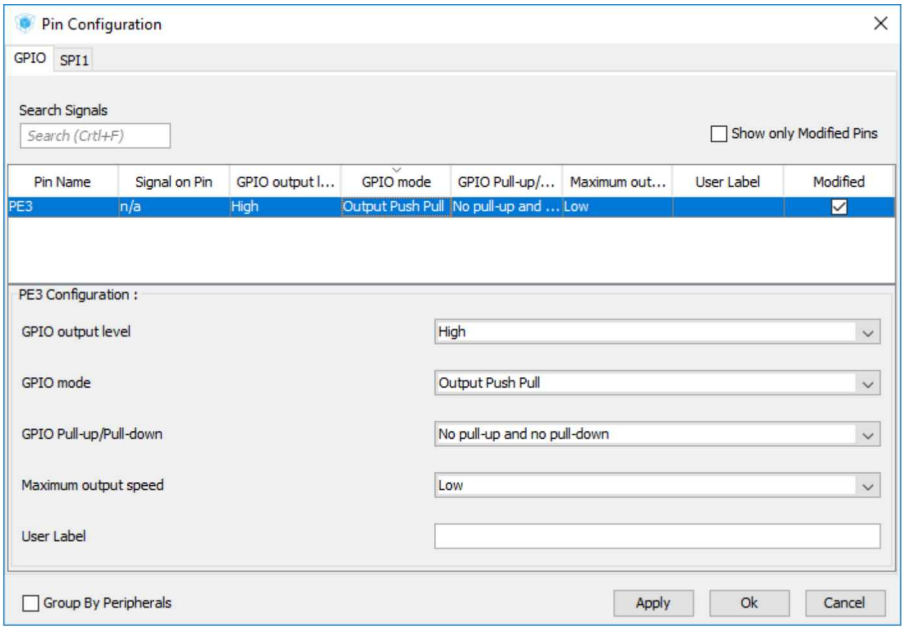
- Konfigurasi CS (*Chip* Select) PE3 sebagai berikut:



- Konfigurasi SPI1 sebagai berikut:



- Pastikan nilai awal CS dalam keadaan High (tidak aktif). Set GPIO sebagai berikut:



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate Code. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Tulis/tambahkan kode berikut pada main.c:

```
uint8_t dataSPITx[2];
int8_t dataSPIRx_X[1], dataSPIRx_Y[1], dataSPIRx_Z[1];

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_SPI1_Init();

  // Jelaskan 5 baris kode di bawah ini
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
  dataSPITx[0] = 0x20;
  dataSPITx[1] = 0x97;
  HAL_SPI_Transmit(&hspi1, dataSPITx, 2, 50);
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
  while (1)
  {
    // Jelaskan 1 baris kode di bawah ini
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);

    // Jelaskan 3 baris kode di bawah ini
    dataSPITx[0] = 0x29 | 0x80;
    HAL_SPI_Transmit(&hspi1, dataSPITx, 1, 50);
    HAL_SPI_Receive(&hspi1, dataSPIRx_X, 1, 50);

    // Jelaskan 3 baris kode di bawah ini
    dataSPITx[0] = 0x2B | 0x80;
    HAL_SPI_Transmit(&hspi1, dataSPITx, 1, 50);
    HAL_SPI_Receive(&hspi1, dataSPIRx_Y, 1, 50);

    // Jelaskan 3 baris kode di bawah ini
    dataSPITx[0] = 0x2D | 0x80;
    HAL_SPI_Transmit(&hspi1, dataSPITx, 1, 50);
    HAL_SPI_Receive(&hspi1, dataSPIRx_Z, 1, 50);

    // Jelaskan 1 baris kode di bawah ini
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);

    HAL_Delay(10);
  }
}
```

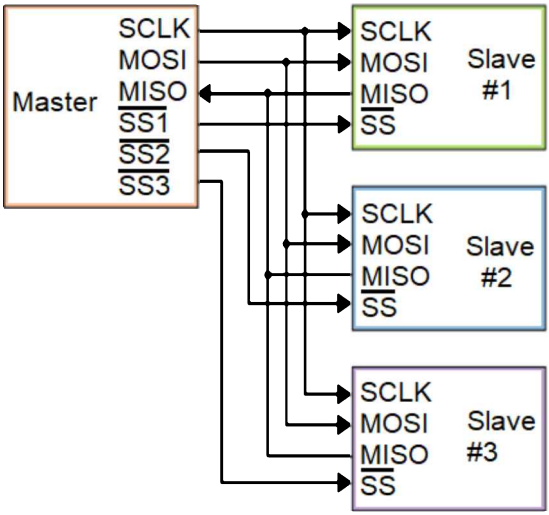
- *Build* dan *load* program.
- *Debug* dan amati nilai dataSPIRx_X, dataSPIRx_Y, dan dataSPIRx_Z. Apa arti nilai-nilai ini?

10.4. Tugas Praktik Bab 10

1) Baca akselerasi pada sumbu Z, ubah ke dalam satuan g, kemudian tampilkan nilainya pada RealTerm. Gunakan skala ±2g dan set *output data rate* pada LIS302DL menjadi 400Hz.
2) Matikan RealTerm, buka program untuk plot grafik, misalnya Arduino Serial Plotter. Gerak-gerakkan (turun dan naik) STM32F407-DISCOVERY sambil dilihat grafiknya pada Serial Plot.

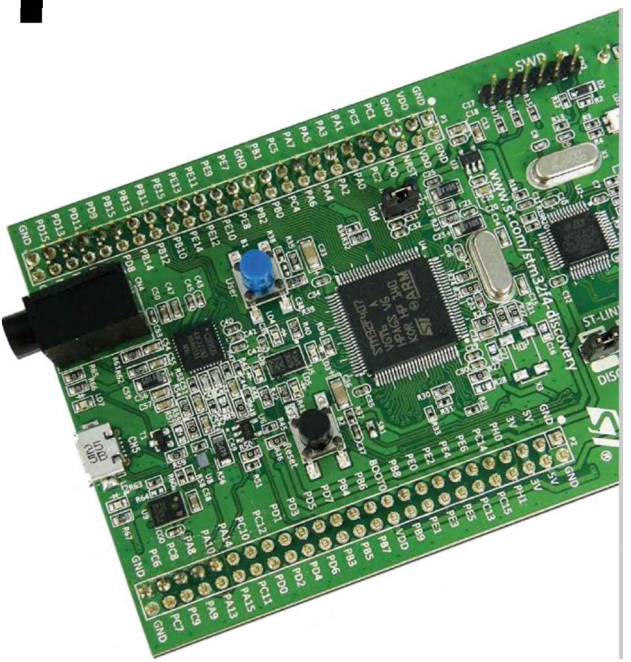
10.5. Latihan Soal Bab 10

- Apa saja keuntungan SPI dibandingkan UART?
- Apa keuntungan topologi SPI Daisy *chain* dibandingkan *independent slave*?
- Jika *clock* SPI yang dihasilkan oleh *master* adalah 1 MHz, maka berapa waktu minimum yang dibutuhkan agar data 8-bit yang dimiliki *Master* terkirim semua ke *Slave* dan data 8-bit yang dimiliki *Slave* terkirim semua ke *Master*?
- Berapa Nilai pin$\overline{SS1}$,$\overline{SS2}$, dan $\overline{SS3}$, sedemikian sehingga *master* dapat berkomunikasi dengan *slave* yang paling atas?



Sistem Embeded Berbasis ARM Cortex-M

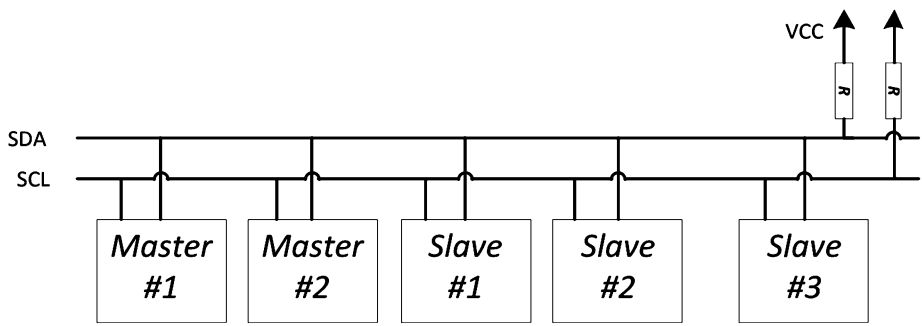
11



Bab 11 I2C

11.1. Prinsip Kerja I2C

Inter Integrated Circuit (I2C) merupakan salah satu komunikasi serial sinkron yang ditujukan untuk komunikasi IC dalam dalam satu PCB (letaknya berdekatan). Kelebihan komunikasi ini adalah simplifikasinya yang hanya memerlukan 2 kabel, yakni satu kabel untuk data dan satu kabel untuk *clock* (Gambar 55). Hal ini sangat membantu menghemat ukuran IC dan PCB, serta mengurangi kompleksitas interkoneksi dalam PCB. Saat ini, I2C sangat banyak digunakan untuk *Interface* modul sensor, EEPROM, LCD, ADC, DAC, dan sebagainya.



Gambar 55. Diagram kabel I2C

Dari segi kecepatan transfernya, I2C dapat dibagi menjadi 3 kategori:

1) *Classic*: 100 Kbits/s
2) *Fast Mode*: hingga 400 Kbits/s
3) *High Speed Mode*: hingga 3,4 Mbits/s

I2C mendukung multi *master* multi *slave*. Setiap *slave* memiliki alamat unik sebesar 7 bit (*slave address*). Setiap berkomunikasi dengan *slave*, *master* harus mencantumkan alamat ini. Komunikasi dapat berjalan dua arah, namun tidak secara bersamaan. Hal ini dinamakan *half-duplex*. *Master* yang menginisiasi komunikasi, membangkitkan sinyal *clock*, mengirim alamat *slave*, dan

menentukan arah dari komunikasi (apakah arahnya dari *master* ke *slave* atau sebaliknya). Sedangkan *slave* merespon *master* hanya jika alamatnya dipilih oleh *master*. Mengingat topologi I2C merupakan multi *master* multi *slave*, maka pin *output* seluruh IC yang terhubung di bus I2C harus diset open drain. Ini dimaksudkan agar tidak terjadi kerusakan IC disebabkan arus yang berlebih karena logika yang berbeda (1 atau 0) antara kedua pin *output*. Dua buah resistor *pull-up* ditambahkan ke *bus* SDA dan SCL.

Setiap *frame* pengiriman ditandai dengan *start bit* dan diakhiri dengan *stop bit.* Tidak ada batasan tertentu mengenai ukuran data yang dikirimkan dalam satu *frame*. Namun demikian, pengiriman dilakukan per 8 bit. Setiap 8 bit selesai dikirim, maka penerima mengirimkan ACK (*akcknowledgment*) untuk menandakan data berhasil dikirim atau NACK (*Not* ACK) jika data gagal dikirim. Perhatikan Gambar 56.

Master menulis ke slave

S	***Slave Address***	***R/W***	*ACK*	***DATA***	*ACK*	***DATA***	*NACK*	***P***

Master membaca dari slave

S	***Slave Address***	***R/W***	*ACK*	*DATA*	***ACK***	*DATA*	***NACK***	***P***

☐ Master ke slave ☐ Slave ke master

R/W: 0=*write*, 1=*read*
S: Bit Start
P: Bit Stop

Gambar 56. Diagram waktu transfer data I2C

11.2. I2C pada STM32F4

STM32F4 memiliki 3 buah I2C dengan fitur dan kemampuan sebagai berikut:

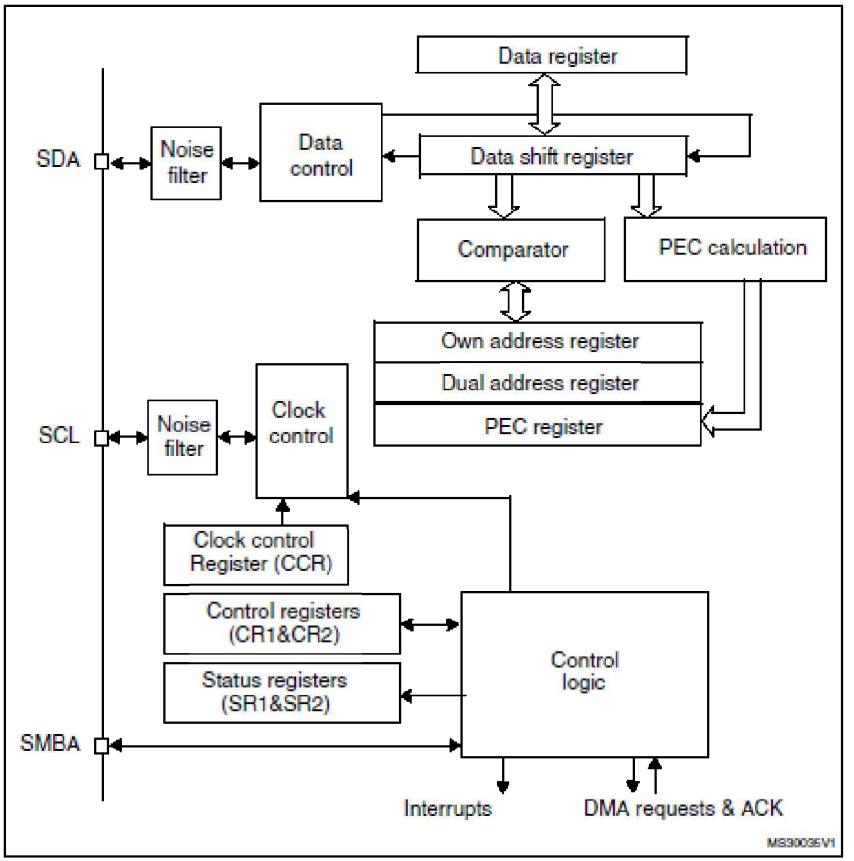
- Kemampuan multi *master*
- Sebagai *master* mampu membangkitkan *clock*, bit start, dan bit stop

- Sebagai *slave* mampu memrogram alamat *slave* (7 atau 10 bit) dan mendeteksi bit stop
- Memilki kecepatan standar (sampai 100 kHz) dan *Fast* (sampai 400 kHz)
- Memiliki filter noise
- Memiliki *flag* yang menandakan *busy*
- Terkoneksi dengan *interrupt* vector. *Interrupt* dapat terjadi baik ketika transfer data sukses maupun ketika transfer data *Error*
- Pilihan I2C menggunakan DMA

Interface dapat beroperasi pada salah satu di antara empat mode berikut:

1) *Slave* transmitter
2) *Slave* receiver
3) *Master* transmitter
4) *Master* receiver

Secara default, STM32F4 berperan sebagai *slave*. Diagram blok I2C pada STM32F4 dapat dilihat pada Gambar 57.



Gambar 57. Diagram blok I2C pada STM32F4

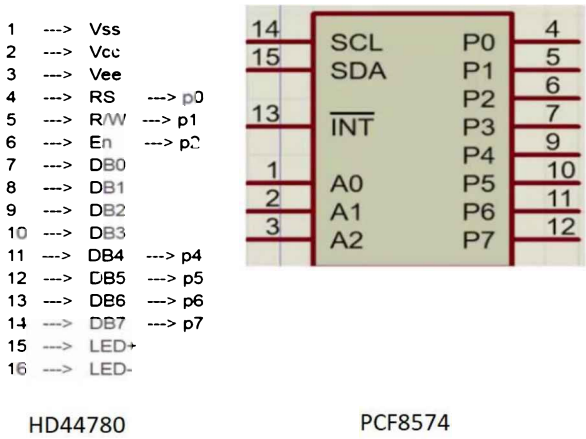
11.3. Praktikum Bab 11

Alat dan Bahan

- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)
- Liquid *Crystal* Display (LCD) HD44780 yang dilengkapi dengan *driver* I2C PCF8574 (Gambar 58)

Gambar 58. LCD HD44780 dengan *driver* I2C PCF8574

Pada praktikum ini, kita mengatur LCD 16x2 menggunakan I2C. HD44780 merupakan LCD dot-matrix 16x2 yang kompatibel dengan MCU 4 atau 8 bit. Tegangan suplainya di antara 2,7 s.d 5,5 V. HD44780 memiliki *Interface* komunikasi parallel sehingga memerlukan banyak pin. Untuk menghemat pin yang digunakan MCU, maka digunakan PCF8574 yang pada dasarnya merupakan expander 8 bit dengan *interface* I2C. Dengan menggunakan PCF8574 ini, MCU hanya menggunakan 4 pin saja (VCC, GND, SDA, dan SCL). Koneksi antara HD44780 dan PCF8574 digambarkan pada Gambar 59.

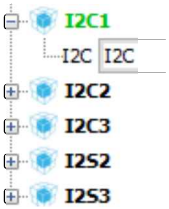


Gambar 59. Koneksi antara HD44780 dan PCF8574

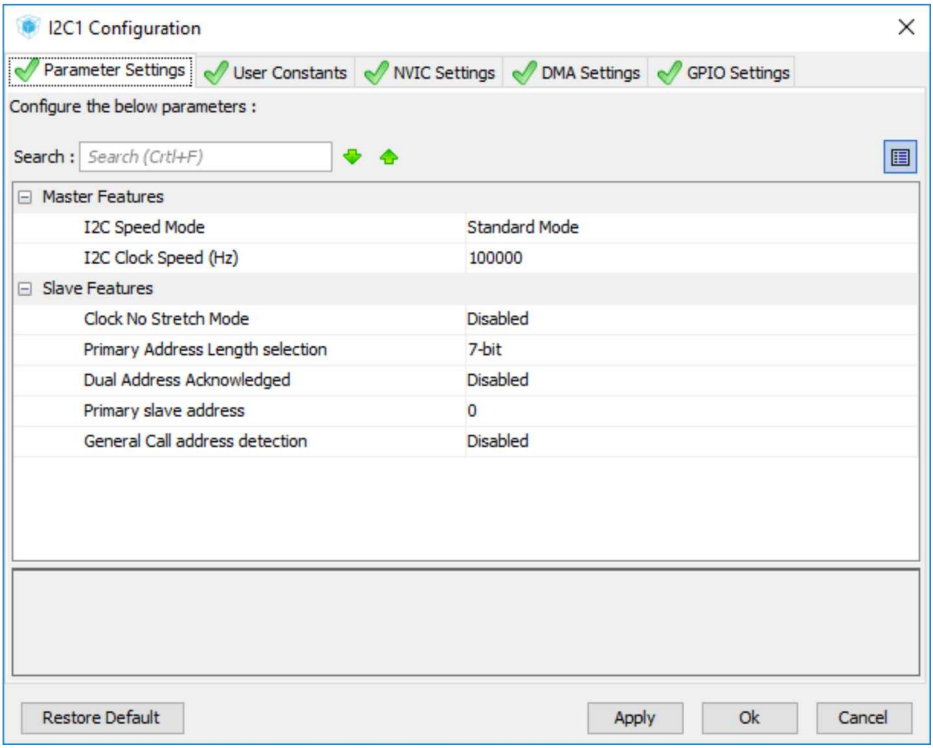
Praktikum 11-01

Pertama-tama, kita akan mendemonstrasikan bagaimana mengetahui alamat dari alat yang terhubung dalam satu jaringan I2C. Kali ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY. Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Gunakan I2C1:



- Konfigurasi I2C1 sebagai berikut:



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate *Code*. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.

- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Tulis/tambahkan kode berikut pada main.c:

```
uint16_t i=0;
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  for (i=0; i < 128; i++)
  {
        if(HAL_I2C_IsDeviceReady(&hi2c1, i, 1, 100) == HAL_OK) break;
  }
  while(1)
  {

  }
}
```

Hubungkan STM32F4DISCOVERY dan LCD sebagai berikut:

STM32F4DISCOVERY	PCF8574
5V	VDD
SCL	SCL
SDA	SDA
GND	GND

- *Build* dan *load* program.
- *Debug* dan amati (watch) nilai i. Nilai i adalah alamat I2C dari LCD. Simpan alamat ini untuk praktikum selanjutnya.

Praktikum 11-02

Setelah mengetahui alamat dari LCD yang terhubung dengan STM32F4, maka pada Praktikum 11-02 ini akan dibuat program untuk menulis string ke LCD. Lanjutkan pemrograman pada Praktikum 11-01 dan lakukan langkah-langkah berikut:

- Buat fungsi lcd_send_cmd(), lcd_send_string(), lcd_send_data(), dan lcd_init() sebagai berikut:

```
#define RS_HIGH  0x01
#define RS_LOW           0x00
#define EN_HIGH  0x04
#define EN_LOW           0x00
#define BACKLIGHT        0x08

void lcd_send_cmd(char cmd);
void lcd_send_string(char *str);
void lcd_send_data (char data);
void lcd_init(void);

void lcd_send_cmd(char cmd)
{
 char data_u, data_l;
 uint8_t data_t[4];
 data_u = cmd&0xF0;
 data_l = (cmd << 4) & 0xF0;

 data_t[0] = data_u | EN_HIGH | RS_LOW; // en=1, rs=0
 HAL_Delay(1);
 data_t[1] = data_u | EN_LOW | RS_LOW; // en=0, rs=0
 HAL_Delay(1);
 data_t[2] = data_l | EN_HIGH | RS_LOW; // en=1, rs=0
 HAL_Delay(1);
 data_t[3] = data_l| EN_LOW | RS_LOW; // en=0, rs=0
 HAL_Delay(1);
 HAL_I2C_Master_Transmit(&hi2c1, 0x4E, (uint8_t *)data_t, 4, 100);
 HAL_Delay(1);
}

void lcd_send_data (char data)
{
 char data_u, data_l;
 uint8_t data_t[4];
 data_u = data&0xF0;
 data_l = (data << 4) & 0xF0;

 data_t[0] = data_u | EN_HIGH | RS_HIGH ; // en=1, rs=1
 HAL_Delay(1);
 data_t[1] = data_u| EN_LOW | RS_HIGH ;// en=0, rs=1
 HAL_Delay(1);
 data_t[2] = data_l | EN_HIGH | RS_HIGH;// en=1, rs=1
 HAL_Delay(1);
 data_t[3] = data_l| EN_LOW | RS_HIGH | BACKLIGHT;// en=0, rs=1
 HAL_Delay(1);

 HAL_I2C_Master_Transmit(&hi2c1, 0x4E, (uint8_t *)data_t, 4, 100);
 HAL_Delay(1);
}

void lcd_init(void)
{
 HAL_Delay(100);
 lcd_send_cmd(0x02);HAL_Delay(1);
 lcd_send_cmd(0x28);HAL_Delay(1);
 lcd_send_cmd(0x0C);HAL_Delay(1);
 lcd_send_cmd(0x80);HAL_Delay(1);
 lcd_send_cmd(0x01);HAL_Delay(1);
}

void lcd_send_string(char *str)
{
 while(*str) lcd_send_data(*str++);
}
```

- Ubah main() sebagai berikut:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();

  lcd_init();
  lcd_send_string("Hello world!");

  while (1)
  {
  }
}
```

- *Build* dan *load* program. Amati layar pada LCD.

Praktikum 11-03

Pada praktikum ini, kita menulis tipe data float menggunakan fungsi sprintf(). Ikuti langkah-langkah berikut:

- Ubah main() menjadi seperti berikut:

```
float fNilai;
char cNilai[16];
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();

  lcd_init();
  lcd_send_string("Hello world!");
  HAL_Delay(2000);

  while (1)
  {
    fNilai = fNilai + 0.25;
    sprintf(cNilai, "%0.3f", fNilai);
    lcd_send_cmd(0x01);
    lcd_send_string(cNilai);
    HAL_Delay(1000);
  }
}
```

- *Build* dan *load* program. Amati layar pada LCD.

11.4. Tugas Praktik Bab 11

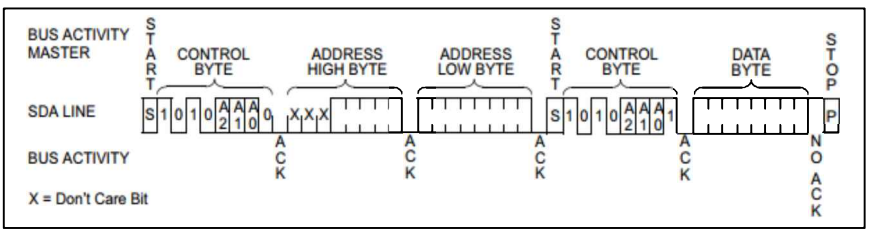
Baca nilai tegangan yang berasal dari potensiometer dan tampilkan nilai tegangannya pada LCD. Atur ketelitian menjadi 2 angka di belakang koma.

11.5. Latihan Soal Bab 11

- Jelaskan maksud kode berikut

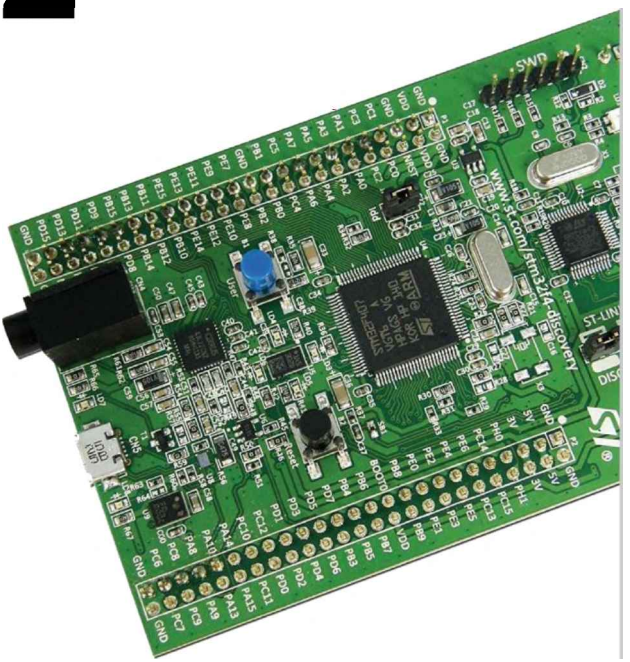
HAL_I2C_*Master*_Transmit(&hi2c1, 0x4E, (uint8_t *)data_t, 4, 100);

- Apa maksud kode lcd_send_cmd(0x02) yang terdapat pada kode praktikum?
- Apa maksud kode lcd_send_cmd(0x28) yang terdapat pada kode praktikum?
- Apa maksud kode lcd_send_cmd(0x0C) yang terdapat pada kode praktikum?
- Apa maksud kode lcd_send_cmd(0x80) yang terdapat pada kode praktikum?
- Apa maksud kode lcd_send_cmd(0x01) yang terdapat pada kode praktikum?
- Di antara UART, SPI, dan I2C, manakah yang mendukung topologi *Master* - *Slave*? (Jawaban dapat lebih dari satu)
- "Pada komunikasi I2C, *master* dapat mengirim dan menerima data, *slave* pun dapat mengirim dan menerima data." Apakah pernyataan tersebut benar?
- Apa yang dimaksud dengan ACK dan NACK?
- Bagaimana cara kita mengetahui I2C address dari sebuah IC?
- Keuntungan I2C dibandingkan SPI antara lain …
- Timing diagram di bawah menunjukkan *Master* I2C sedang *Write* atau *Read*?



Sistem Embeded Berbasis ARM Cortex-M

12



Bab 12 DMA

12.1. Konsep DMA

Direct Memory Access (DMA) adalah fitur pada MCU yang memungkinkan pengaksesan memori tanpa melibatkan CPU sebagai '*middle man*'. Perpindahan memori tidak lagi melibatkan CPU, kecuali hanya di awal dan di akhir prosesnya saja. Proses perpindahan ini kini diatur oleh DMA *Controller*, sehingga CPU dapat melaksanakan fungsi lain yang lebih penting.

Ada 3 cara dalam mengakses I/O, yakni:

1) Polling
2) Interrupt
3) DMA

Polling dan *interrupt* telah dipelajari pada bab-bab sebelumnya. Pada bab ini, dibahas bagaimana mengakses I/O menggunakan DMA. DMA dapat dibagi ke dalam 3 jenis, yakni:

1) Memory-to-memory
2) Peripheral-to-memory
3) Memory-to-peripheral

Langkah-langkah dalam proses DMA adalah sebagai berikut:

1) DMA *Controller* meminta izin kepada CPU untuk mengontrol *bus* data.
2) CPU mengabulkan permintaan DMA tersebut.
3) DMA *Controller* menjawab CPU izin dari CPU tadi.
4) Proses transfer data dilakukan. DMA *Controller* memindahkan data sesuai dengan perintah yang diberikan (*memory*-to-*memory*, *peripheral*-to-*memory*, atau *memory*-to-*peripheral*).
5) DMA *Controller* membuat *interrupt* request untuk memberitahu CPU bahwa transfer data telah selesai.

12.2. DMA pada STM32F4

STM32F4 memiliki 2 DMA *Controller*. Setiap DMA *Controller* dapat menggunakan 8 *stream* dan setiap *stream* terdiri atas 8 channel (CH0

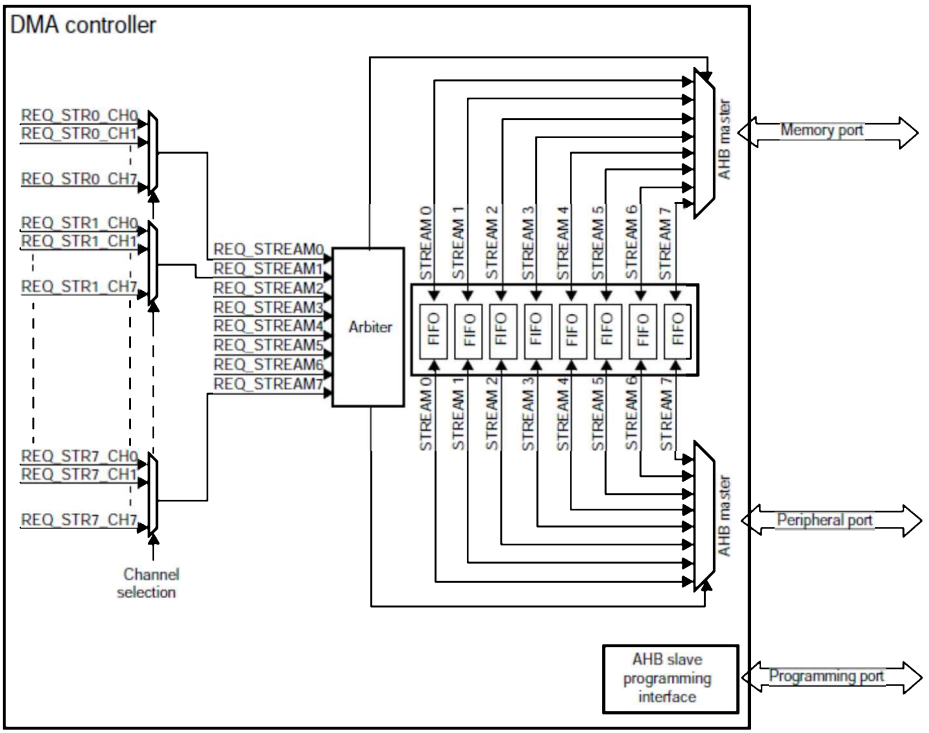
s.d CH7). *Request mapping* dari DMA1 dan DMA2 dapat dilihat pada Tabel 22 dan Tabel 23.

Tabel 22. DMA1 *request mapping*

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX(1)	UART7_TX(1)	TIM3_CH4 TIM3_UP	UART7_RX(1)	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX(1)	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Tabel 23. DMA2 *request mapping*

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A(1)	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A(1)	ADC1	SAI1_B(1)	TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2	SAI1_B(1)	SPI6_TX(1)	SPI6_RX(1)	DCMI
Channel 2	ADC3	ADC3		SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)		USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM



Gambar 60. Diagram blok DMA

12.3. Praktikum Bab 12

Alat dan Bahan

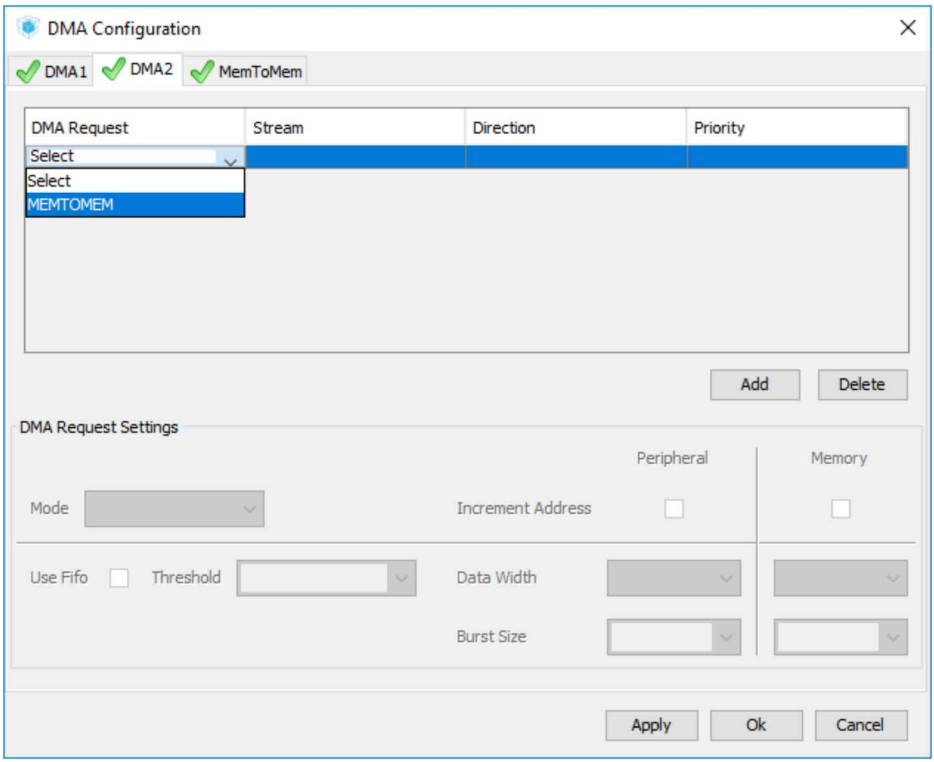
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Praktikum 12-01

Pada praktikum ini, kita akan mendemonstrasikan DMA *Memory* to *Memory*, yakni bagaimana melakukan transfer data dari memori dengan alamat tertentu (*Source* Address) ke alamat yang lain (Destination Address). Kali ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY.

Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Pilih *tab* Configuration, kemudian pilih *Tab* DMA2. Klik tombol Add, kemudian klik Select dan pilih MEMTOMEM. Klik OK untuk menyimpan konfigurasi.



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate *Code*. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Tulis/tambahkan kode berikut pada main.c:

```
uint8_t SrcArray[256];
uint8_t DestArray[256];
int main(void)
{
  HAL_Init();
  for (int i = 0; i<256; i++)
  {
   SrcArray[i] = i;
  }
  SystemClock_Config();
  MX_DMA_Init();
  HAL_DMA_Start(&hdma_memtomem_dma2_stream0,
                (uint32_t) SrcArray,
                (uint32_t) DestArray, 256);
  while (1)
  {
  }
}
```

- *Build* dan *load* program.
- Berikutnya kita akan melakukan *debug* dan melihat efek dari isi memori DestArray sebelum dan sesudah dilakukan transfer data melalui DMA.
- Jalankan mode *debug*.
- Pada window Watch1, masukkan variabel SrcArray dan DestArray.
- Klik kursor pada baris kode yang merupakan eksekusi DMA, yakni HAL_DMA_Start().
- Klik tombol Insert/Remove Break*point*  . Program akan berhenti otomatis di baris kode ini ketika kita mulai *debug*ging.
- Klik tombol Run (atau tekan F5) dan perhatikan isi dari DestArray. Kemudian catat isi DestArray tersebut.
- Klik sekali lagi tombol Run (atau tekan F5) dan perhatikan isi dari DestArray. Kemudian catat isi DestArray tersebut.
- Apa perbedaan isi dari DestArray pada klik tombol Run pertama dan kedua?
- Jelaskan maksud dari kode berikut:

```
HAL_DMA_Start(&hdma_memtomem_dma2_stream0,
              (uint32_t) SrcArray,
              (uint32_t) DestArray, 256);
```

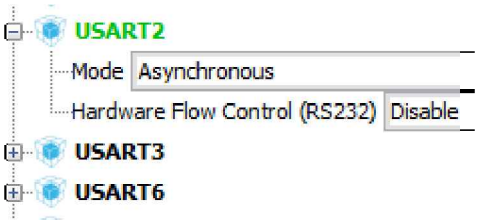
Isi titik-titik berikut:

Fungsi di atas merupakan program untuk … isi dari variabel (alamat memori) … ke variabel (alamat memori) … menggunakan … dengan stream … sebesar … byte.

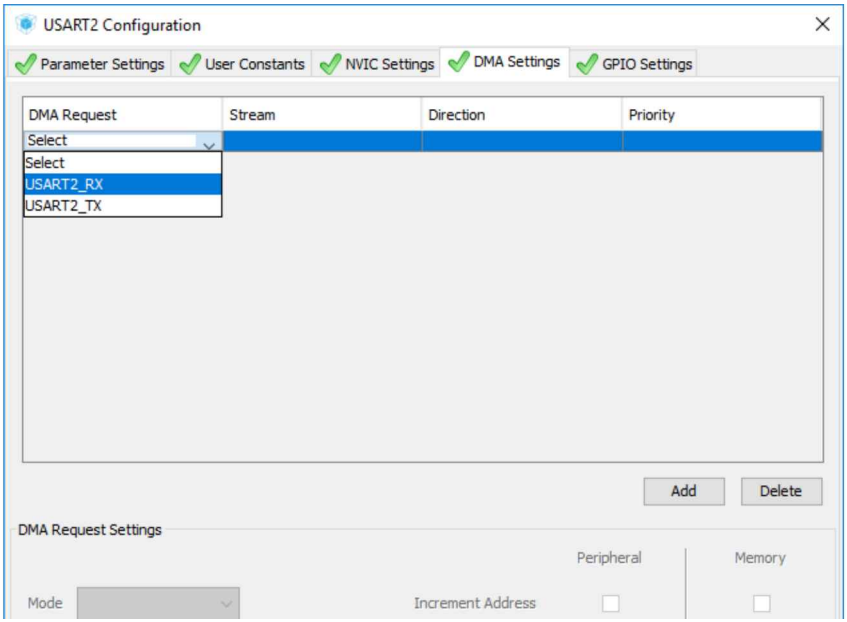
Praktikum 12-02

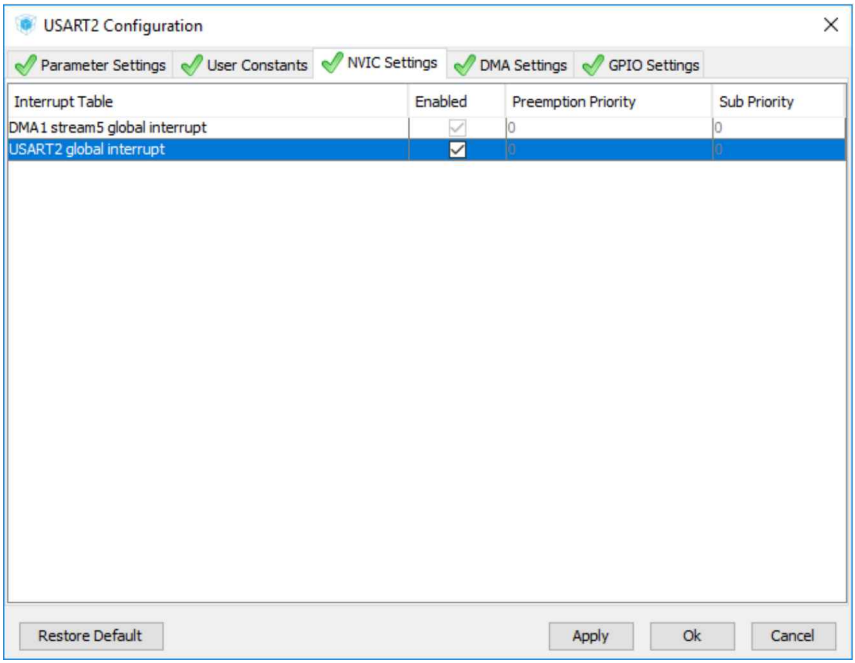
Pada praktikum ini, kita akan mendemonstrasikan DMA *Peripheral* to *Memory*, yakni bagaimana melakukan transfer data dari *peripheral* tertentu (misalnya UART RX) ke memori. Kali ini kita tidak menggunakan parameter *default* pada STM32F4DISCOVERY. Ikuti langkah-langkah berikut:

- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Gunakan USART2:

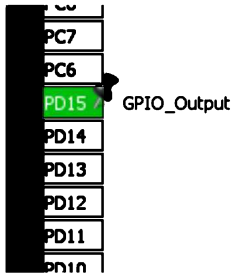


- Pilih *Tab* Configuration, kemudian pilih *Tab* USART2. Klik tombol Add, kemudian klik Select dan pilih USART2_RX. Pada NVIC Settings, aktifkan USART2 global *interrupt*. Klik OK untuk menyimpan konfigurasi.


USART2 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
DMA Request
Stream
Direction
Priority
Select
Select
USART2_RX
USART2_TX
Add
Delete
DMA Request Settings
Peripheral
Memory
Mode
Increment Address


USART2 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Interrupt Table
Enabled
Preemption Priority
Sub Priority
DMA1 stream5 global interrupt
0
0
USART2 global interrupt
0
0
Restore Default
Apply
Ok
Cancel

- Jadikan PD15 sebagai *output*:



- Klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate *Code* dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Tulis/tambahkan kode berikut pada main.c:

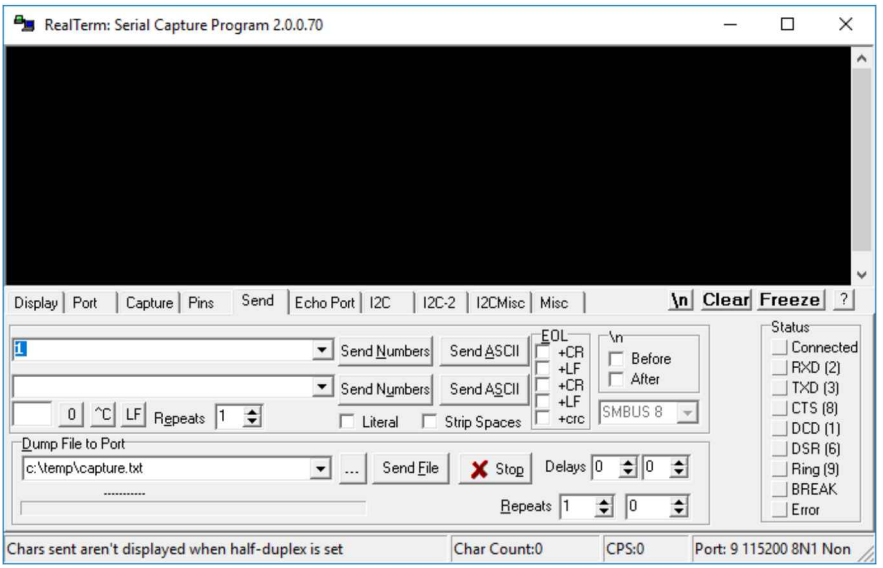
```
uint8_t rx_buff[10];
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  HAL_UART_Receive_DMA(&huart2, rx_buff, 10);

  while (1)
  {
  }
}
```

- Tambahkan callback() sebagai berikut:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
}
```

Build dan *load* program.

- Berikutnya kita akan melakukan *debug* dan melihat efek dari isi memori rx_buff saat dimasukkan nilai menggunakan RealTerm.
- Jalankan mode *debug*.
- Pada window Watch1, masukkan variabel rx_buff.
- Klik tombol Run (atau tekan F5) dan perhatikan isi dari rx_buff.
- Kirim 10 buah karakter dengan menggunakan RealTerm (klik tombol sendNumbers). Setiap kali klik tombol sendNumbers, perhatikan perubahan nilai pada rx_buff.



- Perhatikan LED biru pada board ketika karakter ke-10 dikirimkan. Mengapa demikian? Jelaskan.
- Jelaskan maksud dari kode berikut:

```
HAL_UART_Receive_DMA(&huart2, rx_buff, 10);
```

12.4. Tugas Praktik Bab 12

Buat kode yang menunjukkan transmisi UART dengan menggunakan DMA *Memory* to *Peripheral*.
Data yang dikirimkan adalah:

uint8_t tx_buff[]={0,1,2,3,4,5,6,7,8,9};

Fungsi yang digunakan adalah:

```
HAL_UART_Transmit_DMA(&huart2, tx_buff, 10);
```

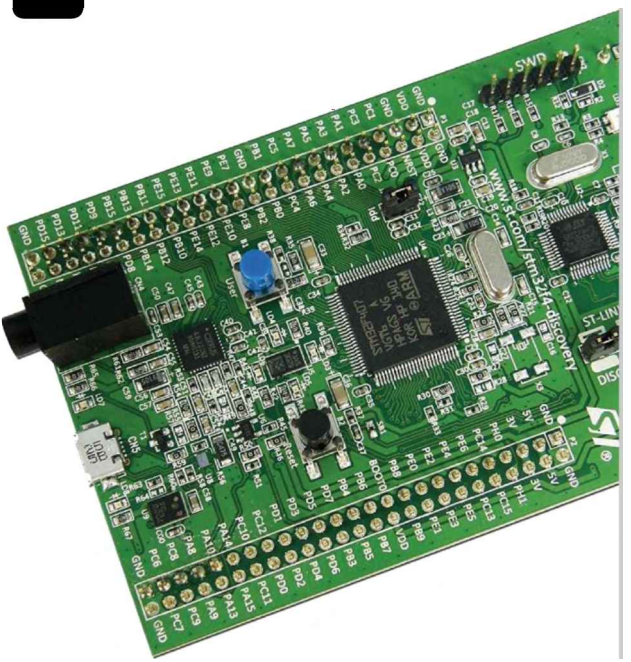
Gunakan RealTerm untuk menerima data.

12.5. Latihan Soal Bab 12

- Jelaskan apa yang dimaksud dengan DMA!
- Sebutkan keuntungan menggunakan DMA!
- Jelaskan langkah-langkah dalam proses DMA!

Sistem Embeded Berbasis ARM Cortex-M

13



Bab 13 RTOS

13.1. Apa itu RTOS

Operating System (OS) merupakan sebuah *Software* yang mengatur seluruh sumber daya yang dimiliki oleh sebuah komputer dan menyediakan layanan untuk program-program komputer yang berjalan di komputer tersebut. Secara umum, OS memiliki 3 fungsi utama, yakni manajemen proses, manajemen memori, dan manajemen I/O. OS sangat banyak tersedia, baik yang berbayar maupun yang gratis bahkan *open source*. Contoh OS antara lain Windows (10, 8, 7, Vista, XP, dan sebagainya), Linux dengan berbagai distronya, Android, Mac OS, MS DOS, dan masih banyak lagi.

Berbeda dengan OS pada umumnya, *Real Time Operating System* (RTOS) ditujukan untuk menjalankan aplikasi *real time* yang memproses data segera setelah data tersebut diterima, tanpa *delay* yang berarti (ada batasan waktu). *Real time* dapat diartikan menyelesaikan *thread* dalam batasan waktu tertentu (*on time*). *Real time* lebih kepada sense manusia ketimbang perhitungan oleh mesin. Ada sangat banyak RTOS yang saat ini banyak digunakan, antara lain FreeRTOS, cocoOS, CooCox CoOS, emboss, iRTOS, Neutrino, BeRTOS, dan sebagainya.

Sebuah MCU seringkali diminta melakukan banyak pekerjaan (disebut juga tugas/*task*/*thread*) dalam satu waktu, sebagai contoh: melakukan konversi ADC, memonitor *timer*, me-generate PWM, memantau sensor, dan sebagainya. Karena *resource* yang terbatas, maka MCU harus melakukan penjadwalan, misalnya mana *thread* yang harus didahulukan dan mana yang harus ditunda. Dengan RTOS, maka MCU Mampu menjalankan beberapa proses (*thread*/*task*) secara "bersamaan".

Karakteristik RTOS antara lain:

- Time constraints
- Consistency
- Reliability

- Predictability
- Performance
- Scalability

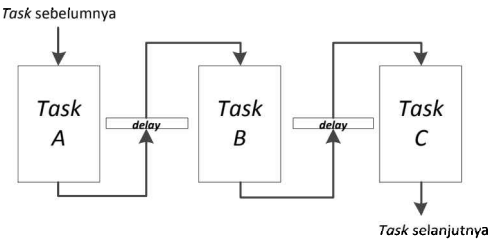
Buku ini menggunakan FreeRTOS, salah satu RTOS yang bersifat *open source* (didistribusikan di bawah lisensi MIT). FreeRTOS merupakan kernel RTOS popular untuk sistem *Embedded* yang telah digunakan oleh sekitar 35 platform MCU.

13.2. Penjadwalan

Penjadwalan (*scheduling*) merupakan fungsi utama RTOS untuk menjamin eksekusi *thread* dilakukan tepat waktu. Ada beberapa jenis penjadwalan, di antaranya adalah:

1) *Coperative* (time sharing)

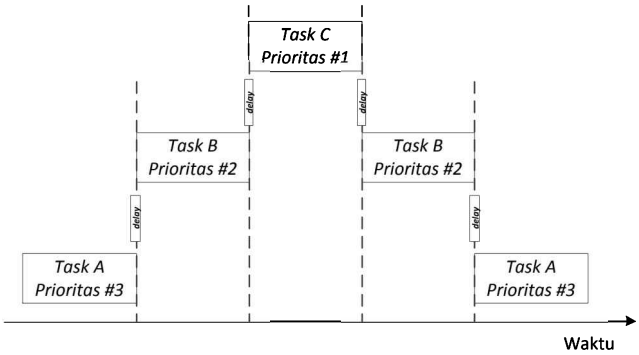
Thread/*task* dikerjakan secara berurutan. Ketika sebuah *task* selesai, maka program beralih ke *task* berikutnya. *Task* tidak memiliki prioritas. Perhatikan Gambar **61**.



Gambar 61. Penjadwalan *Cooperative*

2) *Preemptive* (priority scheduling)

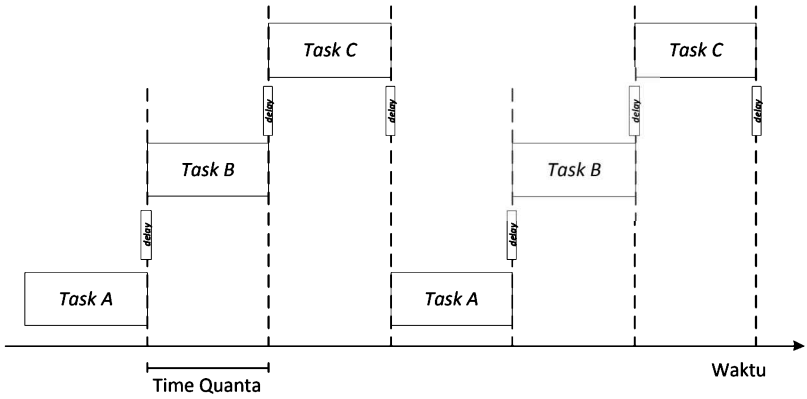
Thread/*task* dijalankan berdasarkan prioritas. Ketika sebuah *task* sedang berjalan, maka dapat diinterupsi oleh *task* lain yang memiliki prioritas lebih tinggi. Perhatian Gambar **62**.



Gambar 62. Penjadwalan Preemptive

3) Round Robin (fixed time slice)

Thread/*task* dijalankan dalam slot waktu yang tetap (*fixed*). Slot waktu ini dinamakan *time quanta*. Ketika *time quanta* habis, maka sebuah *task* dihentikan sementara walaupun belum selesai, kemudian program beralih ke *task* yang lain. Perhatikan Gambar 63.



Gambar 63. Penjadwalan Round Robin

Thread memiliki 3 keadaan (*state*), yakni:

1) Running, yakni *thread* tersebut sedang dieksekusi oleh CPU
2) *Ready*, yakni *thread* tersebut bersiap-siap untuk dieksekusi oleh CPU
3) *Block*ed, yakni *thread* tersebut sedang menunggu *event* tertentu, misalnya I/O

Untuk mengatur penggunaan *resource* (misalnya I/O) yang sama agar tidak diakses secara bersamaan oleh lebih dari satu *thread*, maka digunakanlah *semaphore*. Jika sebuah *thread* mengibarkan *semaphore* terhadap penggunaan suatu *resource*, maka *thread* yang lain tidak dapat mengakses *resource* tersebut sehingga *semaphore* tersebut diletakkan kembali. *Semaphore* memiliki tiga operasi, yakni:

1) *Create*, yakni proses pembuatan *semaphore*
2) *Acquire* (*take*), yakni mengambil *semaphore* dari *thread* lain
3) *Release* (*give*), yakni meletakkan *semaphore* sehingga dapat digunakan oleh *thread* lain

13.3. Praktikum Bab 13

Alat dan Bahan

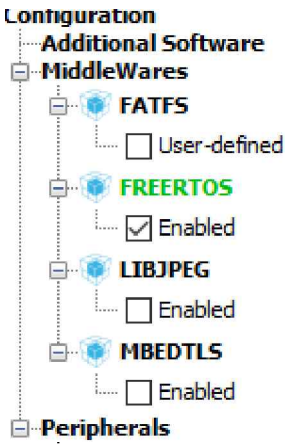
- Komputer yang sudah terinstalasi perangkat lunak Keil µVision V5, STM32CubeMX, dan *Driver* STM32F4DISCOVERY (Virtual COM port)
- STM32F4DISCOVERY (1 buah)
- Kabel USB Mini (1 buah)

Pada praktikum ini, kita akan mendemonstrasikan bagaimana membuat dua buah *thread* sederhana. Kali ini kita tidak menggunakan parameter default pada STM32F4DISCOVERY.

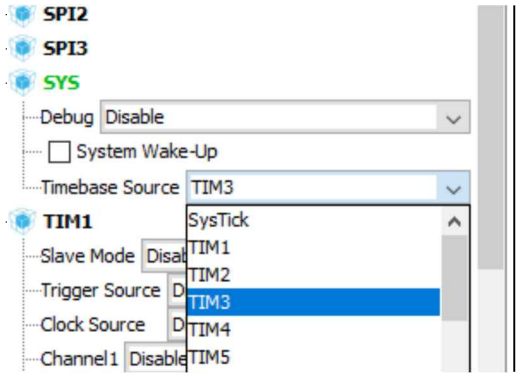
Praktikum 13-01

Ikuti langkah-langkah berikut:

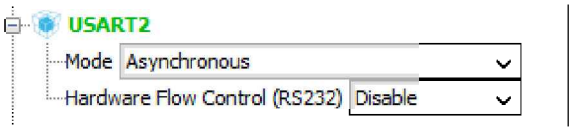
- Jalankan STM32CubeMX, kemudian klik New Project.
- Pilih *tab* MCU Selector. Pada Part Number Search, pilih STM32F407VG. Klik ganda STM32F407VG.
- Aktifkan FreeRTOS.



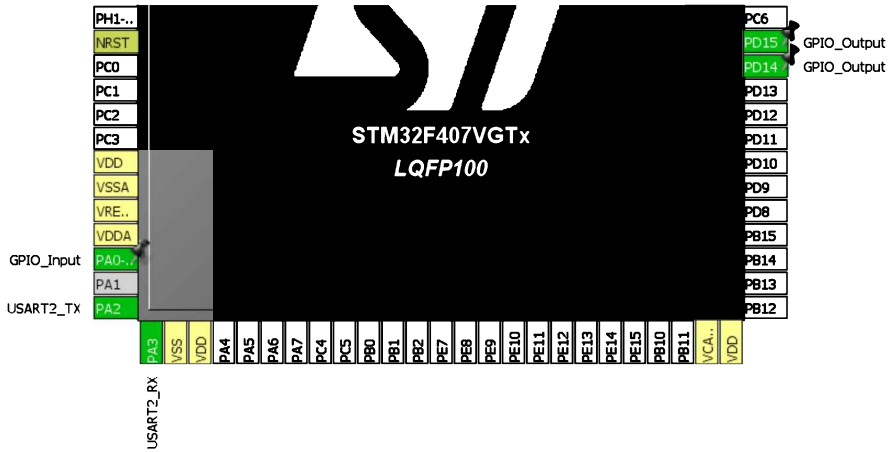
- Pilih *Tab* Configuration, kemudian pilih *tab* SYS. Pilih TIM3 untuk Timebase *Source*. Pada RTOS, usahakan menggunakan *timer* sebagai Timebase *Source*.

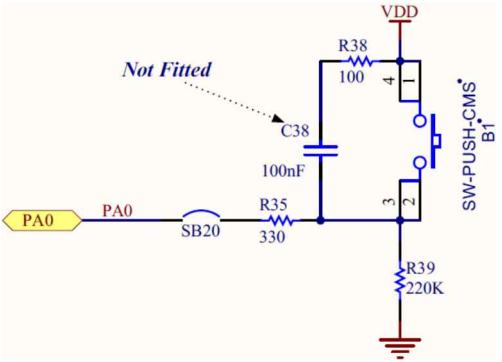


- Aktifkan UART2.

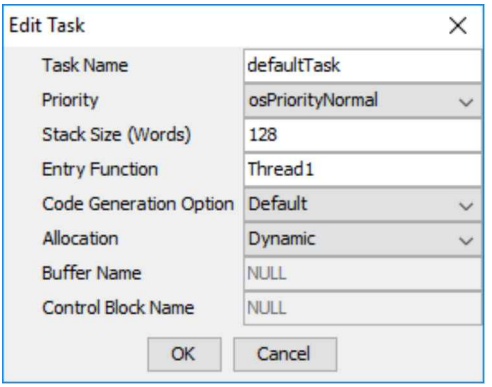


- Jadikan PA0 sebagai *input*, PD15 dan PD14 sebagai *output*. PA0 terkoneksi ke *button* biru. Sedangkan PD15 dan PD14 terkoneksi ke LED.

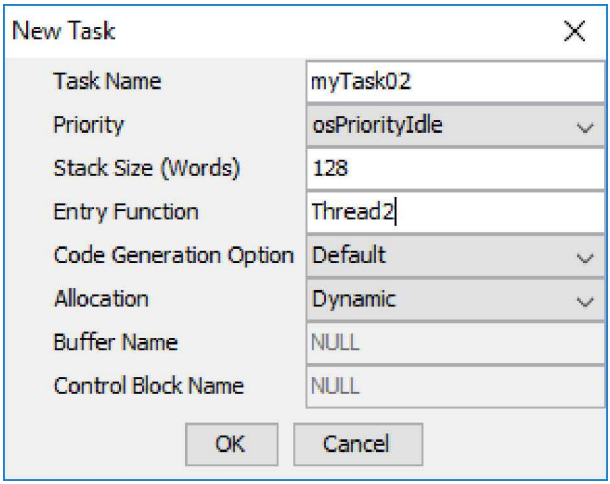




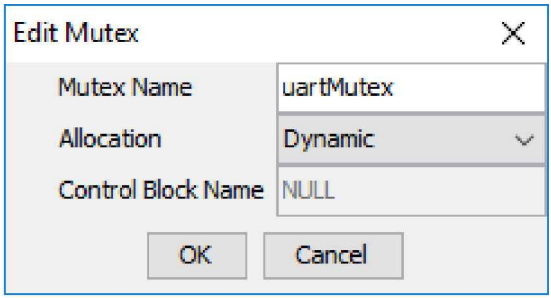
- Klik *tab* Configuration, lalu klik FREERTOS untuk membuka FREERTOS Configuration. Pada FREERTOS Configuration, klik Add . Pada Edit *Task*, ganti Entry Function dengan *Thread*1. Klik OK.



- Klik Add lagi. Ganti Entry Function dengan *Thread*2. Klik OK.



- Buka *tab* Timers and Semaphores. Klik Add pada panel Mutexes. Ganti Mutex Name menjadi uartMutex. Klik OK.



- Konfigurasi STM32CubeMX telah selesai. Sekarang klik Project | Settings.
- Project Settings akan muncul. Isi Project Name dan *folder* sesuai yang diinginkan. Pilih MDK-ARM V5 sebagai IDE. Klik OK.
- Klik Project | Generate Code. dan tunggu STM32CubeMX membuat kode dalam Keil secara otomatis.
- Jika sudah selesai, klik Open Project, maka Keil akan otomatis dijalankan dengan kode yang sudah ada.
- Pada Keil, buka *file tab* main.c.
- Tulis/tambahkan kode berikut pada main.c:

```
/* Thread1 function */
void Thread1(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
    osDelay(500);
  }
  /* USER CODE END 5 */
}

/* Thread2 function */
void Thread2(void const * argument)
{
  /* USER CODE BEGIN Thread2 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
    osDelay(1000);
  }
  /* USER CODE END Thread2 */
}
```

- *Build* dan *load* program.
- Pahami maksud dari kode di atas.

Praktikum 13-02

Ubah kode pada *Thread*1 dan *Thread*2 menjadi sebagai berikut, kemudian jalankan.

```
void Thread1(void const * argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
    osDelay(500);
  }
}

void Thread2(void const * argument)
{
  for(;;)
  {
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET)
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
    }
    else HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
  }
}
```

Pahami maksud kode yang berada di dalam *Thread*2.

Praktikum 13-03

Ubah kode pada *Thread*1 dan *Thread*2 menjadi sebagai berikut, kemudian jalankan menggunakan RealTerm.

```
void Thread1(void const * argument)
{
  uint8_t kirim[10] = "Thread 1\r\n";
  for(;;)
  {
   xSemaphoreTake(uartMutexHandle, portMAX_DELAY);//mengambil semaphore
   HAL_UART_Transmit(&huart2, kirim, 10, 2);
   xSemaphoreGive(uartMutexHandle);//melepaskan semaphore
   osDelay(1000);
 }
}

void Thread2(void const * argument)
{
 uint8_t kirim[10] = "Thread 2\r\n";
 for(;;)
 {
  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0 == GPIO_PIN_SET))
  {
    xSemaphoreTake(uartMutexHandle, portMAX_DELAY);//mengambil semaphore
    HAL_UART_Transmit(&huart2, kirim, 10, 2);
    xSemaphoreGive(uartMutexHandle);//melepaskan semaphore
  }
  osDelay(1000);
 }
}
```

Pahami cara kerja *Semaphore* pada kode tersebut.

13.4. Latihan Soal Bab 13

- Mengapa kita perlu menggunakan RTOS? Apa keuntungannya?
- Jelaskan perbedaan antara penjadwalan Pre-emptive dan Round Robin!
- Jelaskan fungsi *semaphore*! Apa yang mungkin terjadi jika tidak ada *semaphore*?

Daftar Pustaka

Alley, Peter, *Introductory MicroController Programming*, *Master Thesis*, Worcester Polytechnic Institute, 2011.

ARM Architecture Reference Manual, ARM Limited, 2005.

ARM® Cortex®-M4 Processor: Technical Reference Manual (Rev.r0p1), ARM Limited, 2015.

Brown, Geoffrey, *Discovering the STM32 MicroController*, Indiana University Bloomington, 2013.

Conrad, James M & Dean, Alexander G., *Embedded System: An Introduction Using the Renesas RX62N MicroController*, Micriμm Press, 2011.

F. Vahid, T. Givargis, *Embedded System Design: A Unified Hardware/ Software Approach*, 1999.

Fisher, Mark, *ARM Cortex M4 Cookbook*, Packt Publishing, 2016.

Getting Started with Software and Firmware Environments for the STM32F4DISCOVERY Kit, STMicroelectronics, 2016.

Majerle, Tilen. *STM32F4 Discovery Libraries and tutorials for STM32Fxxx series*.

Mittal, Sparsh, *A Survey of Techniques for Designing and Managing CPU Register File*, John Wiley & Sons, 2016.

STM32 configuration and initialization C code generation, STMicroelectronics, 2017.

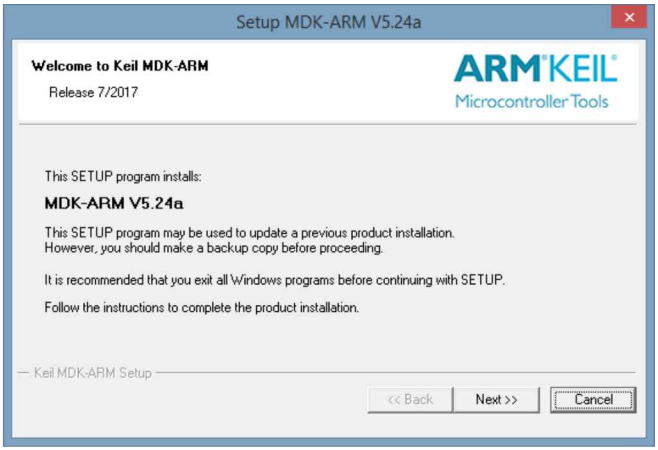
Yiu, Joseph. The Definitive Guide to the ARM Cortex-M0. Elsevier Inc, 2011.

Lampiran

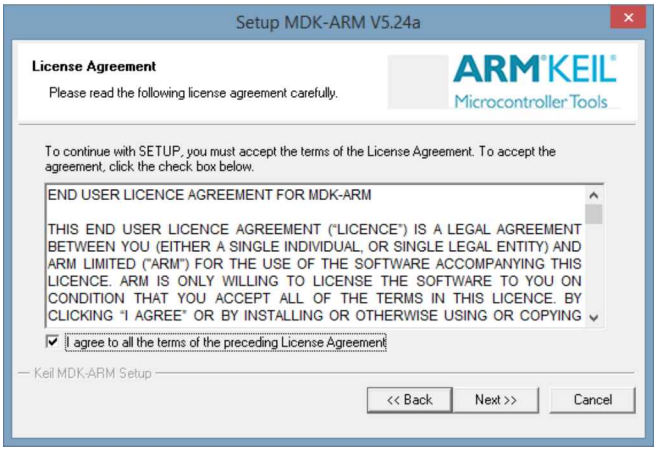
Instalasi Keil uVision

Untuk melakukan instalasi Keil µVision pada komputer, lakukan langkah-langkah berikut.

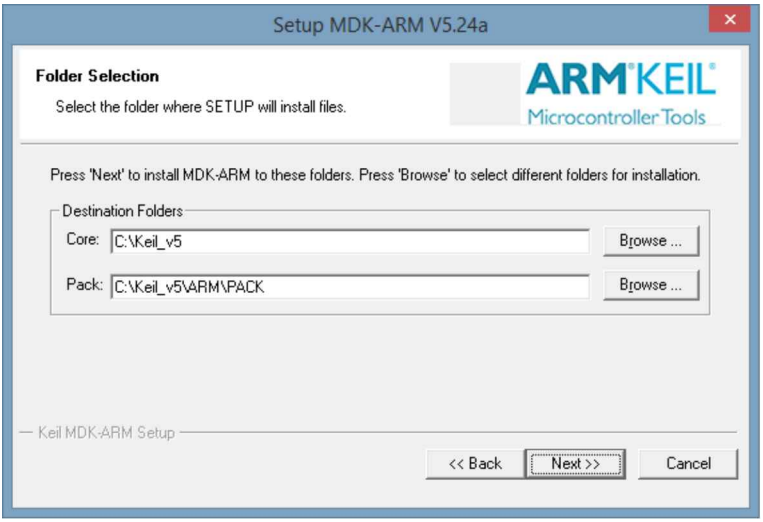
- Klik ganda *file* MDK524a.EXE, maka akan mucul jendela instalasi.
- Setelah jendela instalasi terbuka, klik Next.



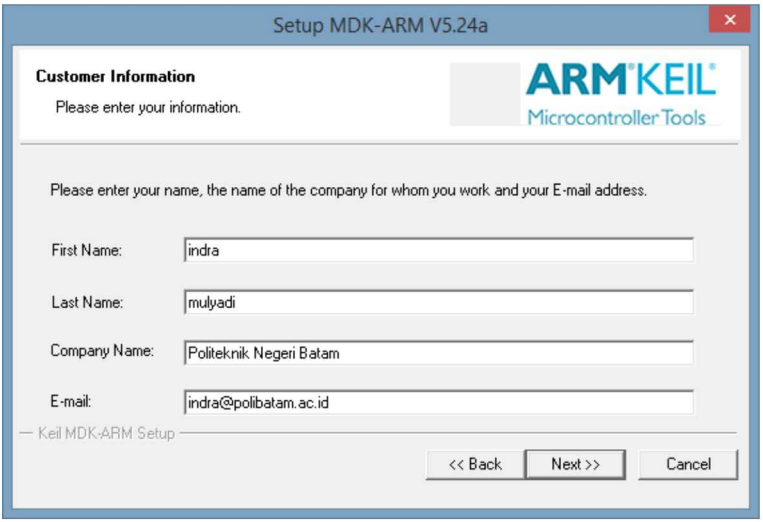
- Centang I agree to all the terms of the preceding Licence Agreement.

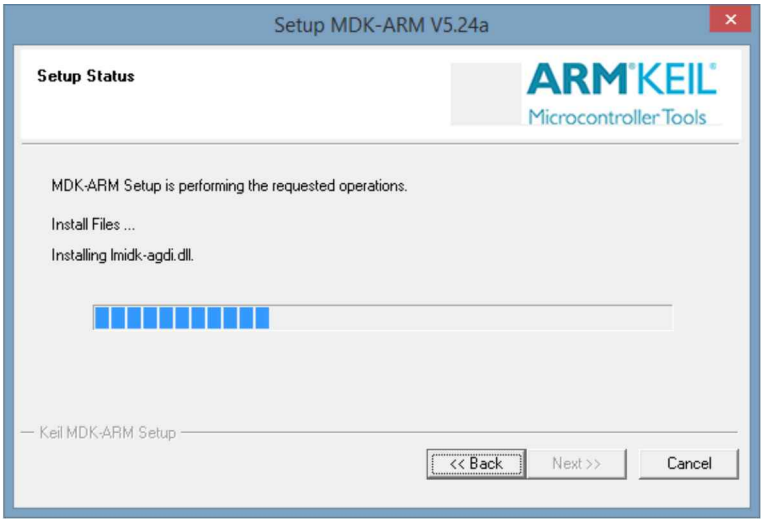


- Pilih *folder* instalasi, kemudian klik Next.

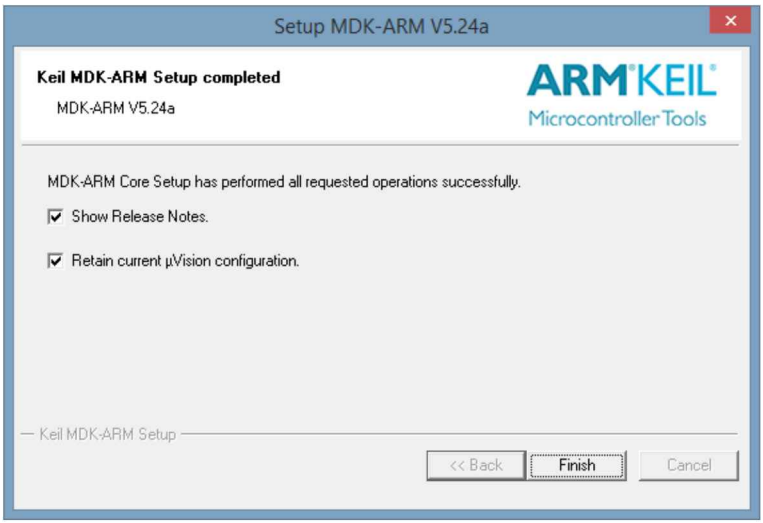


- Masukkan data pribadi (opsional). Klik Next. Tunggu hingga proses instalasi selesai.

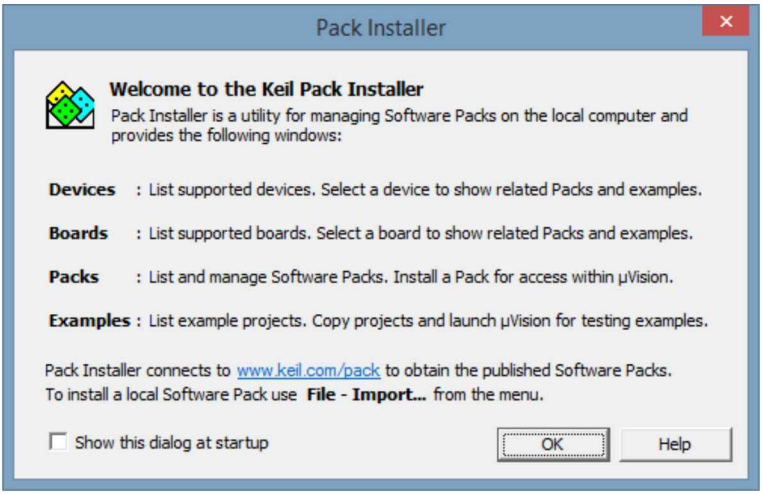


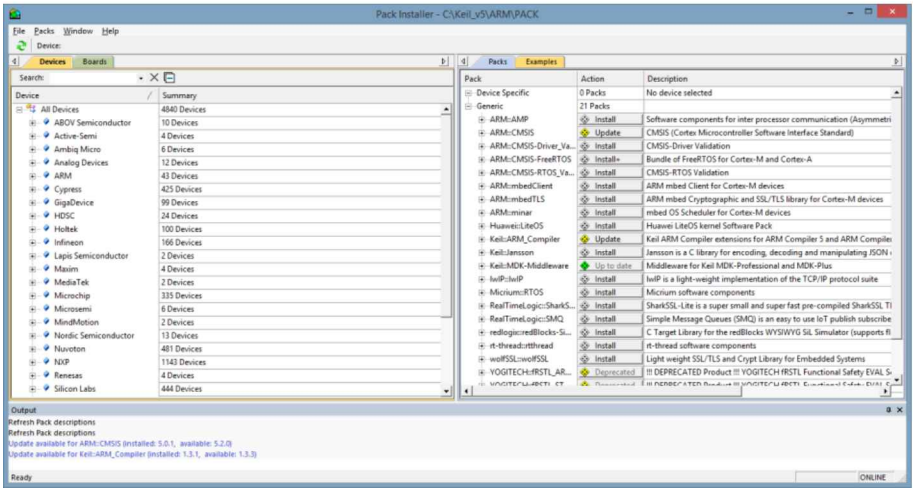


- Klik Finish untuk megakhiri instalasi.



- Jendela Pack Installer akan muncul. Klik OK untuk mengunduh Pack Installer terbaru (Koneksi internet diperlukan pada proses ini). *Pack Installer* ini merupakan utilitas untuk menginstal, memperbarui (*update*) dan menghilangkan *Software Packs* dan dapat dijalankan di dalam Keil µVision ataupun di luar µVision. *Software Packs* itu sendiri dapat berupa *library* dan contoh-contoh proyek.





- Tutup jendela Pack Installer. Proses instalasi selesai. Keil µVision kini sudah siap dijalankan.

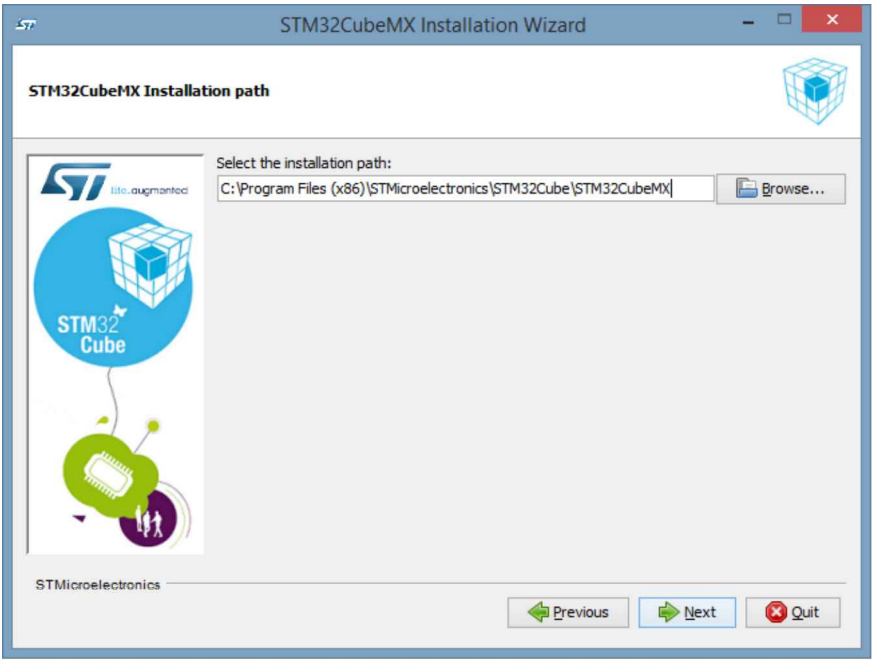
Instalasi STM32CubeMx

Untuk menginstalasi STM32CubeMX, lakukan langkah-langkah berikut.

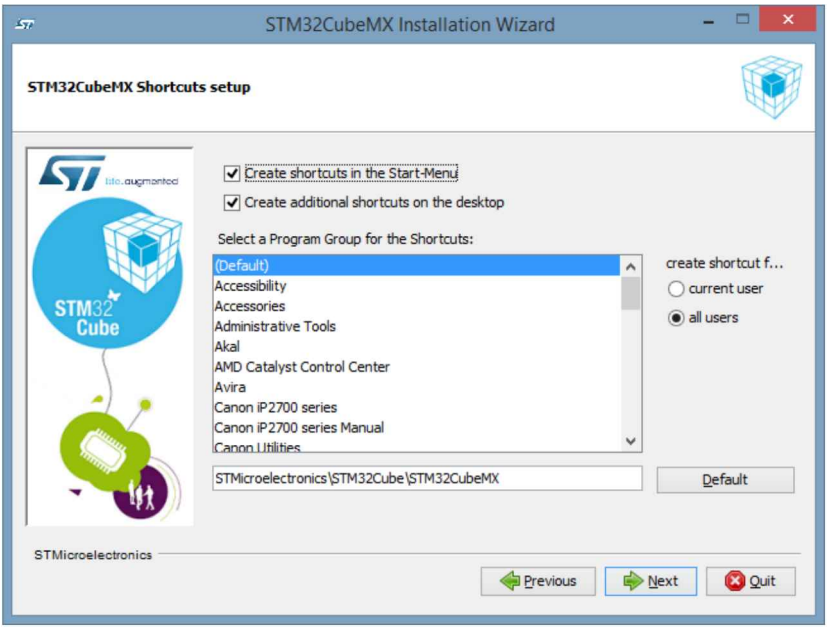
- Ekstrak *file* en.stm32cubemx.zip ke dalam sebarang *folder*.
- Klik ganda *file* SetupSTM32CubeMX-4.23.0.exe, maka akan muncul jendela instalasi.

- Klik Next, kemudian pilih I accept the term of this licence agreement. Lalu klik Next lagi.

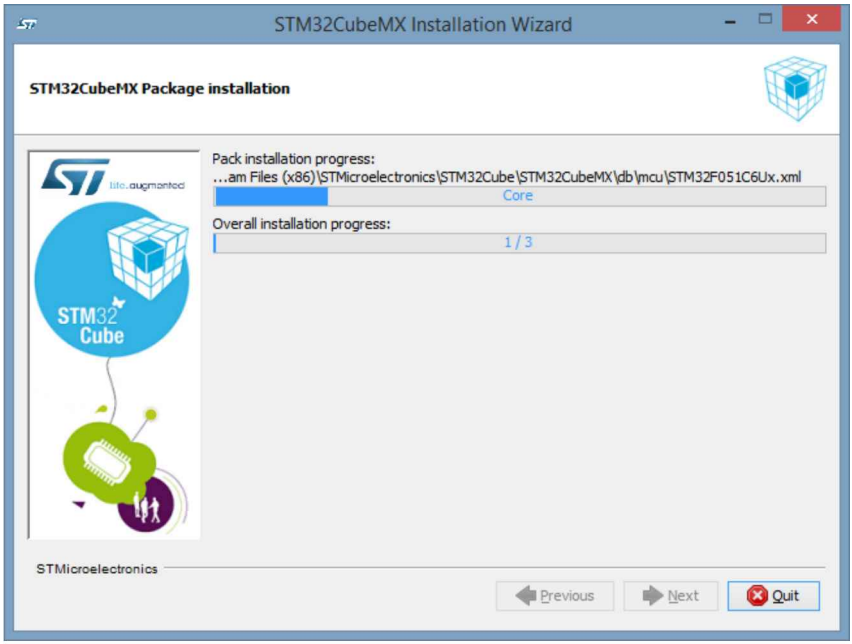
- Pilih *folder* instalasi yang diinginkan, kemudian klik Next.

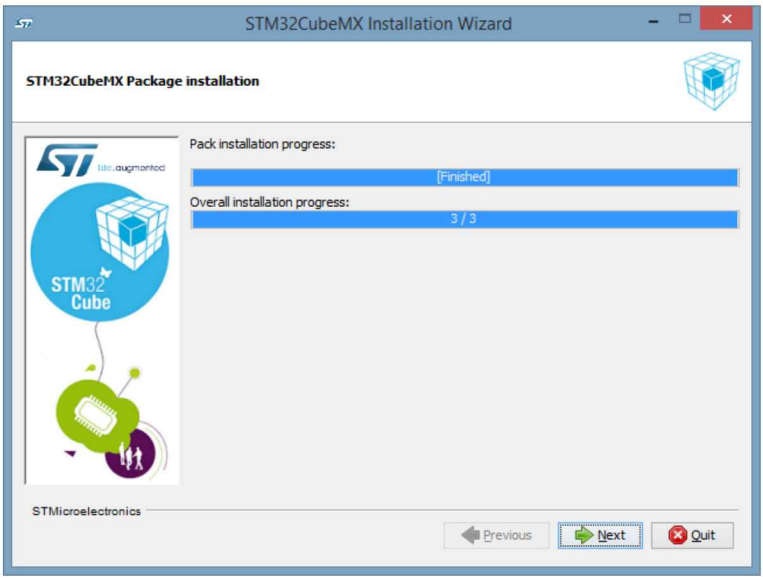


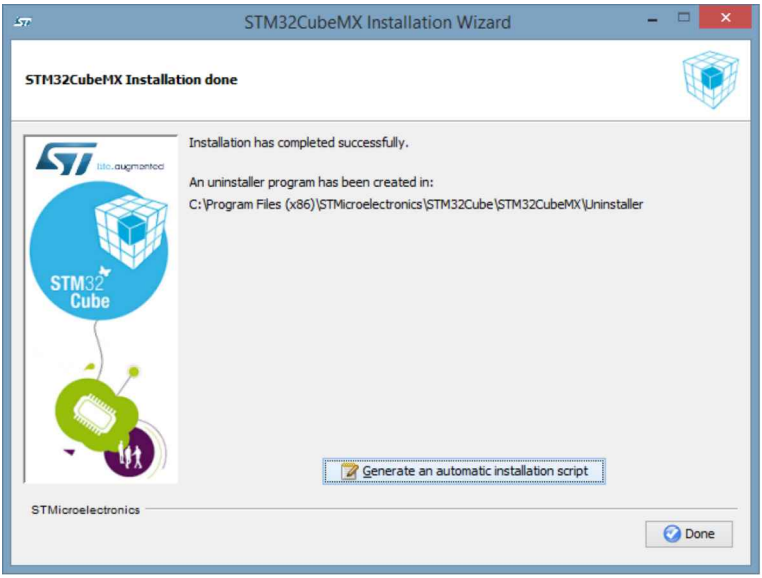
- Pilih default dan klik Next.



- Tunggu hingga proses instalasi selesai. Setelah selesai, klik Next, kemudian klik Done untuk mengakhiri proses instalasi.


STM32CubeMX Installation Wizard
STM32CubeMX Package installation
Pack installation progress:
...am Files (x86)\STMicroelectronics\STM32Cube\STM32CubeMX\db\mcu\STM32F051C6Ux.xml
Core
Overall installation progress:
1 / 3
STMicroelectronics
Previous
Next
Quit


STM32CubeMX Installation Wizard
STM32CubeMX Package installation
Pack installation progress:
[Finished]
Overall installation progress:
3 / 3
STMicroelectronics
Previous
Next
Quit



- STM32CubeMX sudah siap dijalankan.

Indra Hardian Mulyadi lulus pendidikan S1 di jurusan Teknik Elektro, Institut Teknologi Bandung pada tahun 2004 dan S2 di Universiti Teknologi Malaysia pada tahun 2012. Atas beasiswa dari DAAD, mengikuti program PhD Double Degree ke Technische Universität Ilmenau, Jerman pada tahun 2013 hingga 2015. Pernah bekerja di perusahaan bidang manufaktur elektronika, perusahaan avionics, peneliti di Universiti Teknologi Malaysia, dan konsultan beberapa proyek di Singapura, Malaysia dan Indonesia, baik swasta maupun pemerintah. Selain dosen di Politeknik Negeri Batam, saat ini juga sebagai Editor-in-Chief pada Journal of Applied Electrical Engineering.

Eko Rudiawan Jamzuri lahir di Jember, Jawa Timur pada tanggal 15 Maret 1991, meluluskan kuliah Diploma 3 di Politeknik Negeri Batam pada Jurusan Teknik Elektro bidang mekatronika. Kemudian melanjutkan studi Diploma 4 di Institut Teknologi Bandung atas bantuan beasiswa dari kerjasama ITB dan SEAMOLEC. Aktif di Kontes Robot Nasional sejak tahun 2010. Pernah menjadi peserta pada tahun 2010 (KRCI Berkaki) dan tahun 2011 (KRSBI), kemudian menjadi pembimbing tim KRSBI Politeknik Negeri Batam Barelang FC sejak 2013. Saat ini sedang menempuh S2 di National Taiwan Normal University dengan beasiswa dari International Graduate Student Scholarship.

Anugerah Wibisana lahir di Urung, Pulau Kundur, Kabupaten Karimun, Kepulaun Riau pada tanggal 27 April 1994, meluluskan kuliah Diploma 3 di Politeknik Negeri Batam pada Jurusan Teknik Elektro bidang Teknik Komputer. Kemudian melanjutkan studi Diploma 4 bidang Teknik Mekatronika di Politeknik Negeri Batam atas bantuan beasiswa dari kampus Politeknik Negeri Batam. Aktif di Kontes Robot Nasional sejak tahun 2012. Pernah menjadi peserta pada tahun 2012 hingga 2014 pada bidang kontes KRAI (Kontes Robot Abu Indonesia), kemudian menjadi pembimbing tim KRAI Politeknik Negeri Batam dari tahun 2016 sampai sekarang.

Polibatam Press
Politeknik Negeri Batam
Jl.Ahmad Yani, Batam Center, Batam 29461
Tel. 0778-469856
Fax. 0778-463620
press@polibatam.ac.id
http://press.polibatam.ac.id

POLIBATAM PRESS

ISBN 978-602-51056-4-7

9 786025 105647