OULUN AMMATTIKORKEAKOULU

Matti Barsk

Improving timer implementations within a real-time application

Improving timer implementations within a real-time application

Matti Barsk
Thesis work
Spring 2024
Information technology
Oulu university of applied sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Matti Barsk
Title of thesis: Improving timer implementations within a real-time application
Supervisor(s): Kari Jyrkkä
Term and year when the thesis was submitted: Spring 2024
Number of pages: 40

The thesis explored improvements to the timer implementations. The aim was to streamline the source code and improve system performance in a 5G real-time processing application. The study delved into the 5G architecture, application of the Open Event Machine framework, and refactoring methodologies.

The research contextualized the need for improvement by examining the current complexities and redundancies in timer implementations. By removing duplicated functionalities and integrating a single, refined implementation, the thesis proposed a solution that would not only simplify the source code but also enhance its maintainability and testability.

Practical implementations of these improvements involved detailed refactoring of the existing codebase, guided by principles such as readability, maintainability, and minimal impact on existing functionalities. The thesis documented a step-by-step process of the refactoring effort, including the challenges encountered and the strategies employed to overcome them. This process was supported by comprehensive testing regimes that ensured the new implementations met the functional and performance requirements without introducing new faults.

In conclusion, the thesis contributed to the field of information technology by providing a well-documented case study on the application of software development techniques to improve the performance and reliability of critical components in 5G networks. This thesis served as a reference for future projects aiming to optimize software components in real-time applications.

Keywords: 5G Radio Architecture, Physical Layer, Signal processing, Refactoring, Event-driven architecture.

TABLE OF CONTENTS

GLOSSARY

L1	L1 (Layer 1) interface in telecommunications is responsible for physical transmission and reception of signals in a network system.
Data plane	Data plane is part of a network where user packets are transmitted. It is a term to visualize the dataflow of packets through a network infrastructure.
RAN	Radio Access Network
LTE	Long-Term Evolution, technology standard for wireless communication
5G	Fifth generation technology standard for wireless communication
gNB	5G Node B, main component of RAN known as base station.
5GC	5G core network
NR	New Radio
MIMO	Multiple-input and multiple-output. Wireless communication technology that utilizes multiple antennas at both the transmitter and receiver to improve data throughput and signal reliability.
CMake	Open-source cross-platform system for build automation

1 INTRODUCTION

The client of this thesis work is Nokia Networks Oyj, a multinational corporation specializing in telecommunications and information technology. Nokia concentrates its efforts on network infrastructure, software development, and patent licensing. The Oulu site primarily engages in the manufacturing and research and development of base station products. (1.)

In this thesis work 5G radio architecture, open event machine framework and the principles of refactoring will be presented first from a point of view that is relevant for the context of this project. This thesis work itself is about improving source code of timer components, responsible for time critical tasks in the 5G system.

The goal is to remove duplicated functionalities within timer components. This improvement will be done without changing behavior of the source code. Having a single implementation serving all purposes will improve readability of the source code, reduce the complexity of testing, and make the source code more maintainable in the future.

2 5G ARCHITECTURE

The main elements of 5G architecture are presented here to understand the context of where the timers that are under refinement in this work are in the system. Basic elements of the 5G architecture are described in (Figure 1). UE stands for User Equipment, for example a cell phone. RAN (Radio Access Network) is responsible for connecting UEs to 5G core network (5GC). 5GC is the central component of a mobile telecommunications system. It is the backbone infrastructure that handles all the key functions related to call routing, mobility management, session management, authentication, and service provisioning for mobile devices. (2.)

FIGURE 1. Base elements of 5G architecture (2).

2.1 5G Core Network

Generally, in telecommunications a core network is the central infrastructure managing and routing all the data within the network. 5GC utilizes Service-Based architecture, which implements cloud-based design approach. It enables scalability which is required to handle the increasing data traffic in 5G. (2.)

In 5GC architecture framework all architecture elements are defined as network functions (NF). They exist either in control plane or user plane. Control plane functions handle network management and control signaling like handovers, device authentication and management of the paths that the network data packets will travel. User plane on the other hand is responsible for the actual transmission of user data packets in the network. (2.)

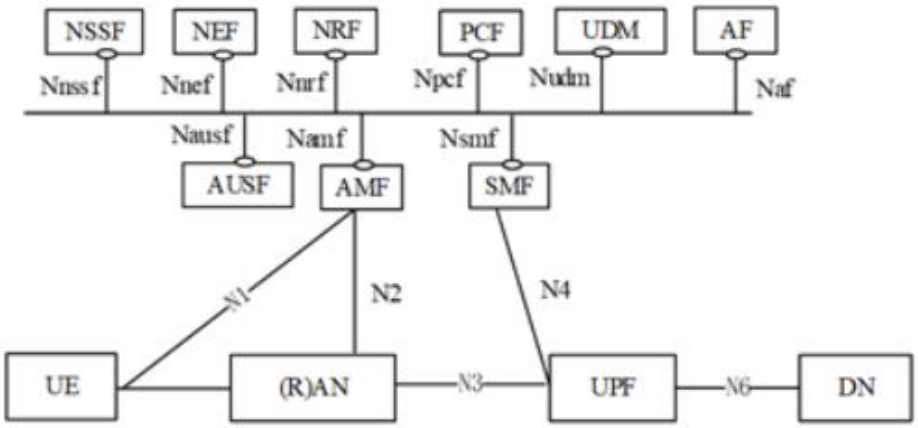


FIGURE 2. 5G core network architecture simplified (2).

AMF (Access Management Function) is a part of the core processing in the control plane. It manages the access of UEs within the 5G network and makes sure that UE connects to the appropriate service and network in the 5G environment. (2.) AMF receives Non-Access Stratum (NAS) messages from the UE through RAN. NAS messages are registration, authentication and security related protocols that enable RAN and AMF to manage UEs access in the network. (2.)

UPF (User Plane Function) handles routing of the user data packets within the user plane and ensures high speed data transmission between UE and external networks. With RAN it forwards user data packets between UE and the core network. UPF manages Quality of Service (QoS) for user data traffic. QoS is responsible for measuring and routing packet flow in a way ensuring that quality requirements are met. (2.)

2.2 Radio Access Network

5G RAN consists of radios, antennas, and baseband units (gNB). Together they cover a single land area (cell) and multiple cells form the access network which enables user equipment (UE) to connect to the core network and vice versa. (15). Connection from gNB to UE is known as downlink and a connection from UE to gNB as an uplink. RAN: s purpose is to connect UEs to the core network (5GC). (3.)

RAN carries user data into the core network. All processing in the user plane is separated into layers. These layers form a protocol stack (Figure 4). Protocol stack layers are PHY, MAC, RLC and in case of user plane function PDCP-U, SDAP and finally UPF. Control plane protocol stack is similar (Figure 4) (7.)

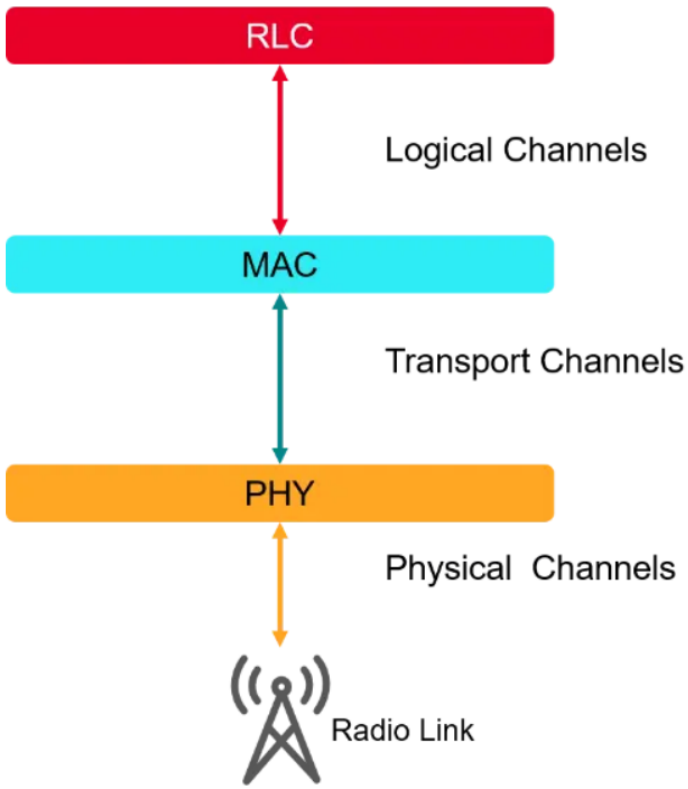


FIGURE 3. Protocol stack channel types (7).

Data between layers flows through channels shown in (Figure 3). As an example, initial access and RRC connection which will be described in the next chapter is processed on the channels of the PHY and MAC layer.

PHY (Physical Layer)

Communicates directly with user equipment through physical channels by modulating and demodulating radio signals in over the air communication (3).

MAC (Medium Access Control)

Manages access to shared radio resources between multiple users. It handles resource allocation, prioritization of traffic, and control signaling. (8.)

RLC (Radio Link Control)

Ensures that data transmission over radio link is error free by detecting and correcting errors in the packet flow (8).

PDCP-U and C (Packet Data Convergence Protocol)

PDCP provides its services to RRC and upper user plane layers. These services include compression and decompression of packets and headers, encryption and decryption of user data, and integrity protection and verification of control plane data, these services enable secure delivery of data. (8.)

RRC (Radio Resource Control)

Responsible of establishing, maintaining, and releasing connection between UE and gNB depending on the situation. Tasks such as connection setup and mobility management of radio bearers and control signaling for handovers are handled in RRC. (3.)

SDAP (Service Data Adaptation Protocol)

Provides Quality of Service management and data flow mapping to specific radio bearers. Ensures that all types of traffic receive appropriate QoS parameters. (8.)

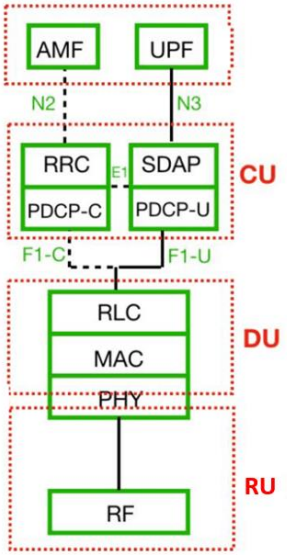


FIGURE 4. 5G Protocol stack simplified. In the picture F1-C is Control plane and F1-U is User plane protocol. (20).

RU is the lowest level of the architecture and is responsible for receiving, transmitting, amplifying, and digitizing the radio signals. Through antennas RU interfaces directly with the physical radio environment. The DU and CU known as computational functions transmit the digitized signal into the network for further processing. (13.)

Functional split affects how processing load of protocol stack is shared between RU, DU, and CU. Functional splits will have an impact on performance and can be switched for different use cases. (14.) Figure 4 describes one way to split functions between layers.

2.3 Initial Access

To establish and maintain radio connection between UE and RAN following interactions happen. When UE moves to new coverage area, it performs initial access procedure to gain access to the network. During it UE and gNB will synchronize downlink and uplink transmission to enable communication between each other and the 5G network. (12.)

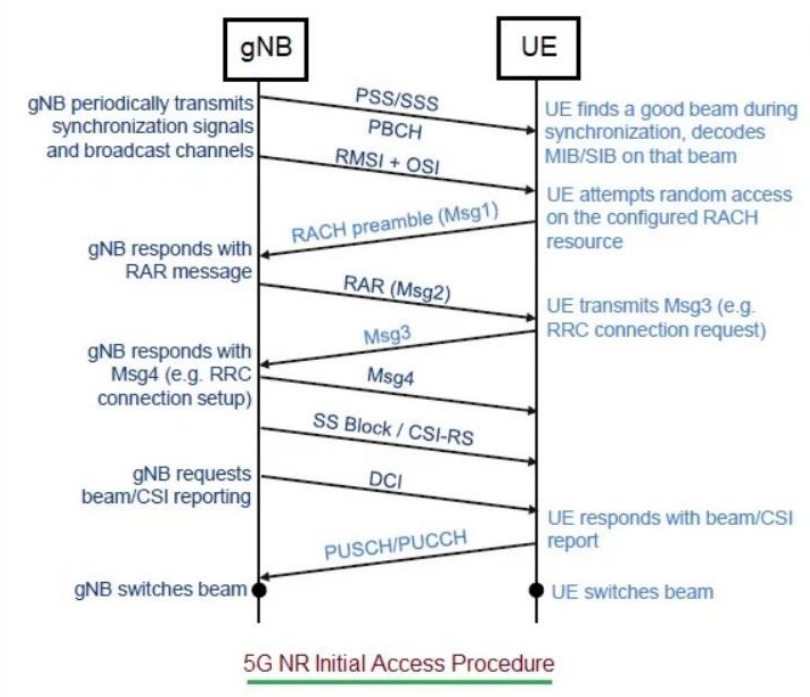


FIGURE 5. Initial access procedure (12).

Downlink synchronization

PSS (Primary Synchronization Signal), SSS (Secondary Synchronization signal) and PBCH (Physical Broadcast Channel) are referred as an SS-Block (SSB). Before attempting initial access UE measures the beam SSB is being transmitted on and finds the one with the strongest signal strength. (12.)

PSS and SSS are transmitted periodically in the downlink from gNB to UE for obtaining the cell identity and frame timing (Figure 5) (3). This is required for aligning the transmission and reception timing correctly within the network. PBCH channel as a part of SSB in the downlink carries MIB (Master Information Block) which allows UE to decode SIB1 (System Information Block 1). SIB1

contains cell access and selection related information and defines scheduling other SIB blocks. (12.) This process can be referred as downlink synchronization.

Uplink synchronization

After selecting a suitable cell UE initiates RACH (Random Access Procedure) procedure which consists of 4 messages exchanged between UE and gNB. UE uses PRACH (Physical Random-Access Channel) to transmit random access preamble to gNB. A preamble is a unique identifier for UE and serves as a request for gNB to allocate resources to UE for further communication (Figure 5). (3.)

When gNB receives the preamble transmission (Msg 1) from PRACH channel it sends a response message to UE which is called random access response (Msg 2). Msg 2 contains following information: time advance to adjust UEs timing correctly, RAPID (Random Access Preamble ID) matching the preamble UE have sent, initial uplink access grant for UE, and a RA-RNTI (Random Access - Radio Network Temporary Identifier). (3.)

With the initial uplink access granted by gNB UE transmits RRC connection request (Msg 3) on uplink PUSCH (Physical Uplink Shared Channel) (12). This message contains information about establishment cause and UEs identity (3).

Contention resolution (Msg 4) is gNB:s response to Msg 3 sent by UE. It confirms that the gNB has identified UE correctly and provides UE with C-RNTI (Cell Radio Network Temporary Identifier). C-RNTI is a unique identifier for the specific UE to differentiate it from other users while communicating with the gNB after RACH procedure. (3.)

When RRC connection is established, and downlink and uplink are configured UE and RAN can start transferring user data packets over the air interface. (3.)

2.4 Physical layer (L1)

The physical layer of wireless communication serves as the foundation for transmitting and receiving data over the air interface. Its key functions include modulation, demodulation, channel estimation, forward error detection, MIMO antenna processing, and beamforming. Supporting these functions is a structured framework including frame structure, numerology, and resource grid allocation. (3.) These concepts are presented to understand physical layer signaling on a symbol level.

2.5 Physical Channels

Communication between the base station and user devices involve multiple physical channels for both uplink and downlink transmission. Physical channels are used to carry user data and control information over the air interface. Following channels each serve their own purposes in data transfer and control signaling in both directions. (3.) Some channels might already be familiar from the initial access chapter.

2.5.1 Downlink

PDSCH (Physical Downlink Shared Channel)

This physical channel carries user data and control information from gNB to UE. PDSCH shares capacity on time and frequency basis. PDSCH has an adaptive modulation. Depending on the link quality it can use either QSPK, 16-QAM, 64-QAM, or 256-QAM modulation. (4.)

PDCCH (Physical Downlink Control Channel)

This physical channel carries only L1 downlink control information (DCI). Mostly it uses QSPK-modulation (4). Its primary purpose is to schedule downlink transmissions on PDSCH and the uplink transmissions on the PUSCH (5).

PBCH (Physical Broadcast Channel)

Physical broadcast channel carries the necessary system information to enable UE to access 5G network (4). It is the primary channel through which a base station transmits Master Information Block (MIB) to UEs. MIB is the minimum requirement for UE to connect with the cell. (3).

2.5.2 Uplink

PRACH (Physical Random-Access Channel)

Is used in the uplink for initial access requests from the users (4). Whenever a new user tries to connect to the 5G network PRACH is initialized first to gain initial access, which in this context means a process where UE and gNB (5G network) acquire uplink synchronization and the UE obtains specified ID for radio access communication. (3.)

PUSCH (Physical Uplink Shared Channel)

PUSCH is the uplink counterpart for PDSCH therefore it carries the user data in the uplink direction. Depending on the link quality it can use either QSPK, 16-QAM, 64-QAM, or 256-QAM modulation. (4.)

PUCCH (Physical Uplink Control Channel)

Carries UCI (Uplink Control Information) when there is no PUSCH allocation. (4.)

2.6 5G Frame structure

5G frame structure refers to how radio resource elements are organized within a time-frequency grid in the 5G wireless communication system. It is formed of key components which are slot structure, resource grid and numerology. 5G frame structure is more flexible than its predecessor to support new bandwidths and varying subcarrier spacings between them. (4.)

Numerology

5G NR introduces different numerologies which represent different subcarrier spacing configurations in an Orthogonal Frequency Division Multiplexing (OFDM) system. OFDM is a commonly used modulation technique in wireless communication systems. It divides the available bandwidth into multiple narrowband subcarriers, each carrying a small partition of the total data. For example, radio signal (Carrier) is set to frequency spectrum of 2.4GHz. Within the carrier signal we allocate specific frequency ranges inside the bandwidth to individual subcarriers each operating at slightly different frequencies within the spectrum. (3.)

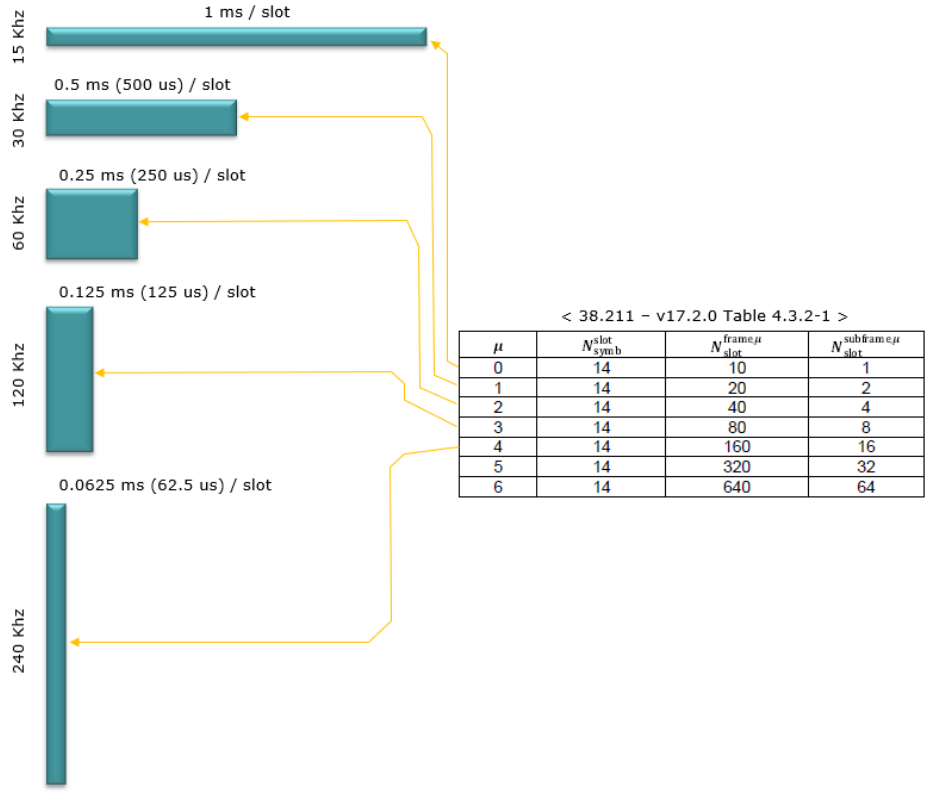
When comparing 5G to LTE the noticeable difference is that LTE has only one subcarrier spacing (15kHz) and in 5G NR subcarrier spacing varies from 15kHz to 960kHz. Wider subcarrier spacing enable faster data transmission and lower latency because the symbol duration is shorter. Smaller spacing does not perform as fast but has increased coverage. Figure 6 describes that numerology 0 has a subcarrier spacing of 15kHz which is used by LTE. Numerology 1 with 30kHz subcarrier spacing is typical for 5G NR at cmWave frequency band, and the numerologies from 2 to 4 with their spacings ranging from 60 kHz to 240kHz are used on larger bandwidth available on higher mmWave frequencies. (3.)

Parameter / Numerlogy (u)	0	1	2	3	4
Subcarrier Spacing (Khz)	15	30	60	120	240
OFDM Symbol Duration (us)	66.67	33.33	16.67	8.33	4.17
Cyclic Prefix Duration (us)	4.69	2.34	1.17	0.57	0.29
OFDM Symbol including CP (us)	71.35	35.68	17.84	8.92	4.46

FIGURE 6. Numerologies from 0-4 (3).

Regardless of what numerology is in use, these measurement units are always the same. Radio frame which is 10ms and subframe that is 1ms. Cyclic prefix is a period between transmitted symbols. It protects from interference of previously transmitted symbols. With normal Cyclic Prefix one

slot contains 14 OFDM symbols. With extended Cyclic Prefix one slot contains only 12 OFDM symbols. Numerologies influence single slot duration (Figure 7). As subcarrier spacing increases slot duration decreases, which means that more slots fit into a single radio frame (10ms) as described in (Figure 8). (3.)



< 38.211 – v17.2.0 Table 4.3.2-1 >

μ	N_{symb}^{slot}	$N_{slot}^{frame,\mu}$	$N_{slot}^{subframe,\mu}$
0	14	10	1
1	14	20	2
2	14	40	4
3	14	80	8
4	14	160	16
5	14	320	32
6	14	640	64

FIGURE 7. Slot duration in different numerologies (3).

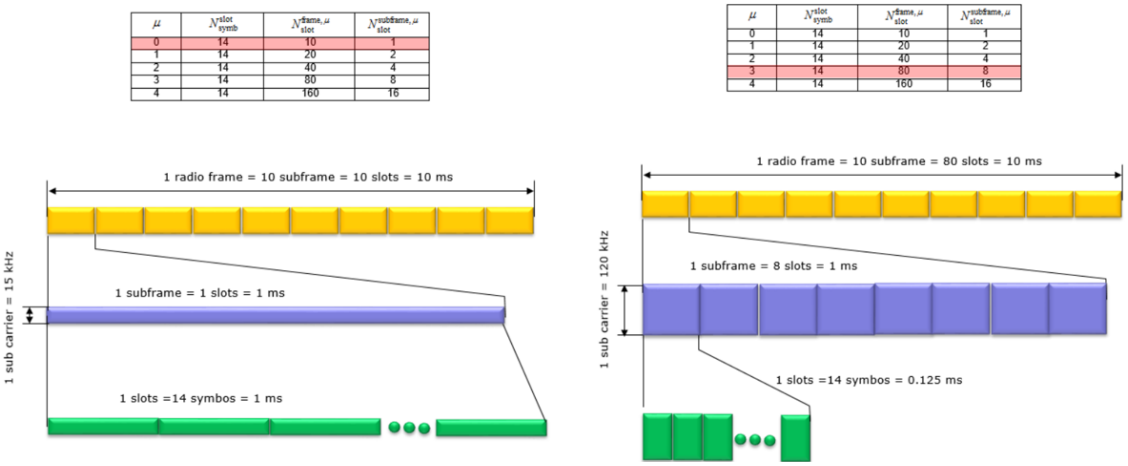


FIGURE 8. Comparison on slot duration and symbol amount in a single radio frame between numerologies 0 and 3 (3).

2.6.1 Resource grid

Resource grid is a time and frequency presentation on the allocation of physical resources in the wireless spectrum (Figure 9). Resource grid allows to track signals on multiple frequencies in time domain. One subframe in the time domain (y-axis) and full carrier bandwidth in the frequency domain (x-axis). One rectangle on this grid is a resource element. One resource element represents one subcarrier during one symbol. Twelve consecutive subcarriers in the frequency domain are equal to one resource block. (3.)

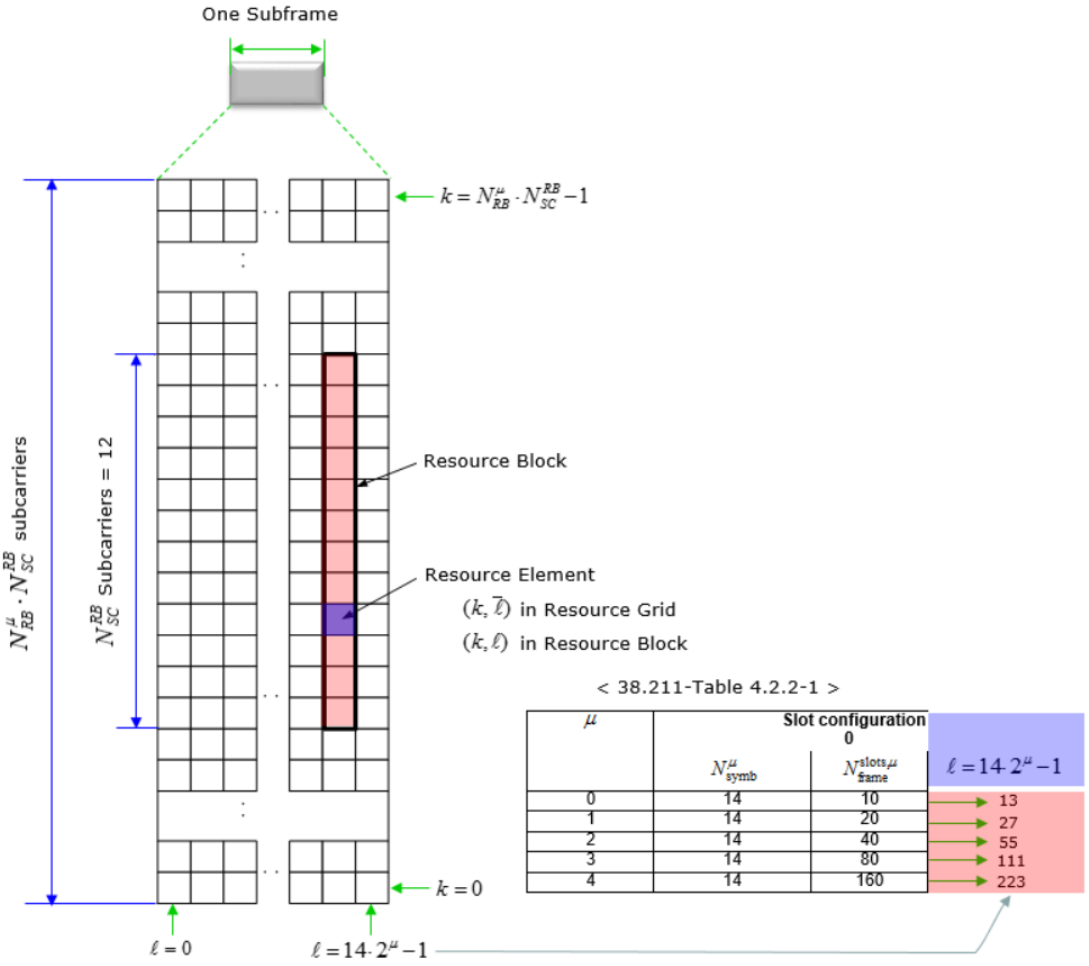


FIGURE 9. Resource grid. y-axis is frequency (carrier), and x-axis is time (one subframe = 1ms) (3).

2.6.2 TDD (Time Division Duplex)

TDD is a spectrum usage technique in mobile networks. TDD is more common in 5G networks. In TDD both uplink and downlink use the same frequencies on a spectrum, but at different times. In comparison FDD (Frequency Division Duplex) transmits uplink and downlink at the same time on different spectrums. TDD and FDD both have their own frame structures. In case of TDD symbols can be configured either uplink, downlink or flexible. In FDD mode downlink and uplink are two separate bands which are always available and separated by guard band to avoid interference. TDD uses single band that is divided into time slots (Figure 10). These time slots are divided into uplink and downlink periods. Uplink and downlink phases are separated from each other with guard periods. This is done to reduce interference between slots. (4.)

In 5G TDD is used in spectrum ranging from 2.5GHz all the way to 90GHz and above. FDD is only used for spectrum below 3GHz frequencies. In 5G the low FDD bands are used for wide area coverage, reliability, and deep indoor penetration. Higher spectrums like 2.5-4.9GHz are used for higher data rates. Spectrum above 24GHz is used for extreme data rates in local hotspot areas. (4.)

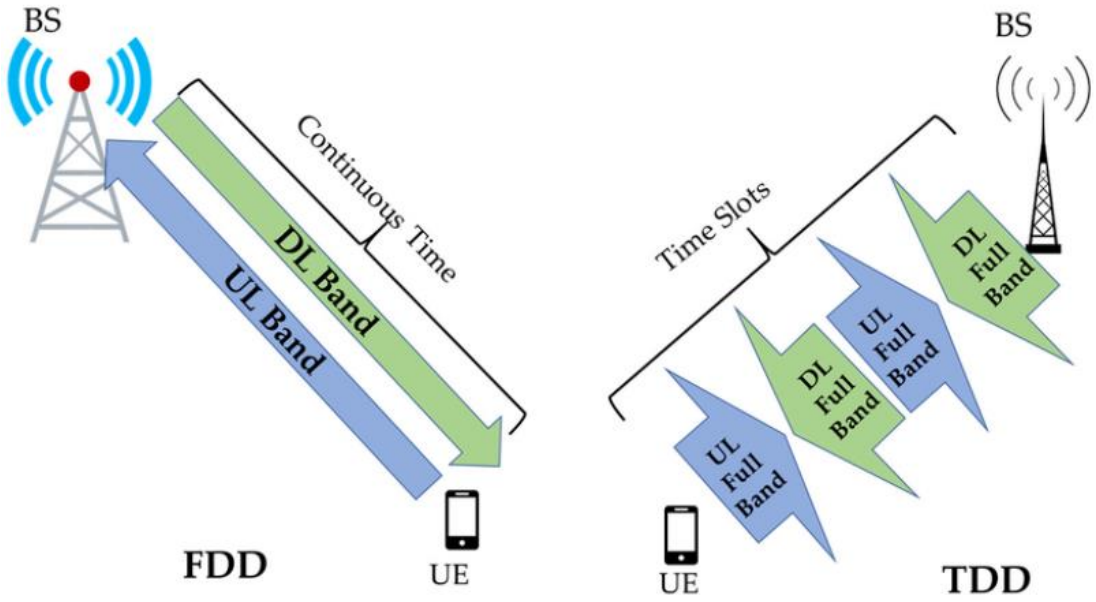


FIGURE 10. FDD and TDD comparison (9).

2.7 Modulation

Modulation is used to encode digital data into a wireless signal. All data on physical channels is modulated. Modulation defines the amount of information that one resource element can carry (Figure 9). The more complex the modulation is, the more information a single resource element can carry. Complex modulation requires a good link quality. More complex modulation achieves higher speeds but can cause interference and data corruption. (17.)

Constellation is a visualization for modulation (Figure 11). A constellation is a grid of points that each have a digital value assigned on them. Variations of frequency and amplitude are used to communicate a specific coordinate on the constellation (17). Horizontal axis is called in-phase "I" and vertical axis is "Q" quadrate.

Representation of 16-QAM modulation (Figure 11). It has 4x4 constellation points, which means that one resource element contains 4 bits of information.

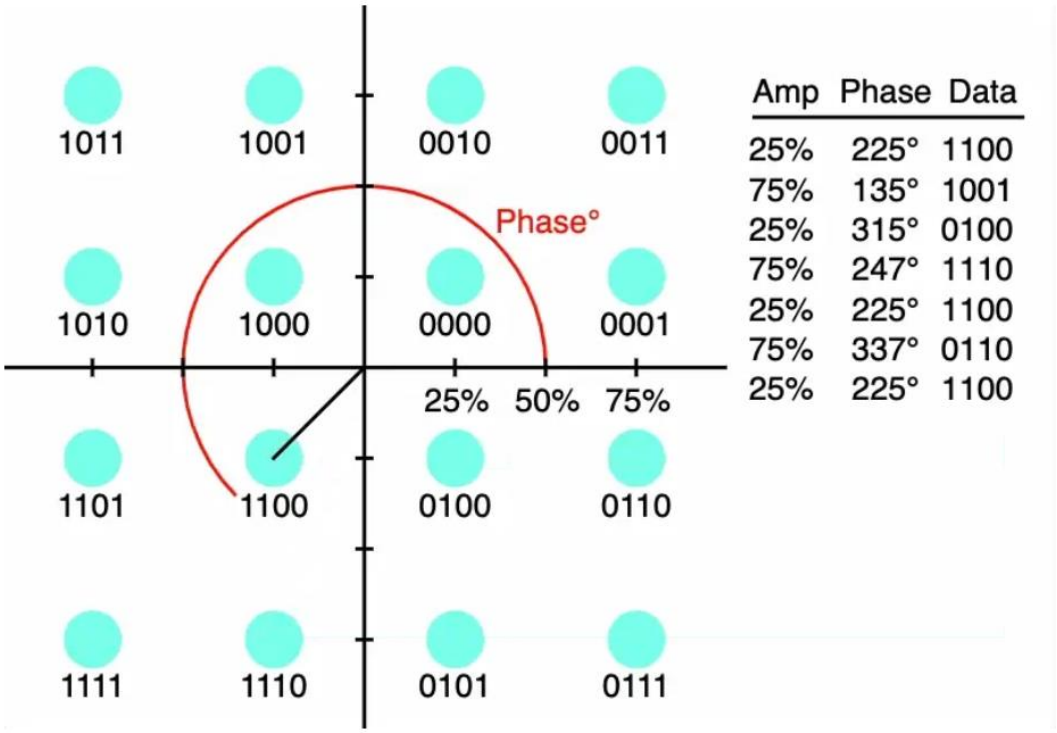


FIGURE 11. A constellation grid horizonal axis-I and vertical axis- Q (17).

Phase Shifted Key Modulation (PSK) and Quadrature Amplitude Modulation (QAM) modulation techniques are commonly used in cellular communication. PSK adjusts only phase of the signal and amplitude stays constant. QAM adjusts phase and the amplitude of a signal. (17.) QAM combines two amplitude modulated waves into a single frequency to increase bandwidth. The two carrier waves on the same frequency are out of phase with each other by 90°. This state is known as quadrature. This also forms the constellation diagram. (23.)

With its better spectral efficiency QAM modulation is more common in 5G. Since QAM modulation adjusts phase and amplitude more constellation points can be fitted into a single resource element. With more constellation points higher speeds can be achieved (Figure 12). (17.) 5G uses 16-QAM, 64-QAM and 256-QAM modulations.

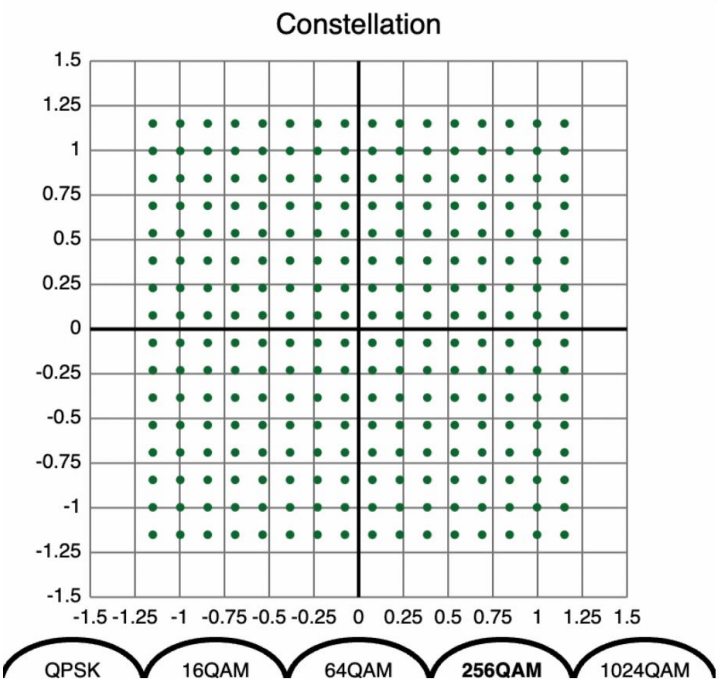


FIGURE 11. 256-QAM modulation one RE carries 8 bits of information (17).

On the receiving end the digital data on the signal transmitted is demodulated. Demodulation works like modulation but in reverse. The receiver knows which constellation point the received signal belongs to by comparing the I and Q components with the reference constellation points and selecting the closest match. Reference constellation points are predefined and known by the transmitter and receiver. (24.)

2.8 Timers

Timers offer a way to track time, schedule tasks and manage asynchronous operations. They are useful in delays, operations that need a timeout before execution, or when a certain action needs to be executed at a fixed time period. (10.)

Because a mobile network is a time critical system, precise timing of events in the radio interface is required. As explained in the theory part, bandwidth is separated into multiple frequency channels and each of them are further divided into timeslots. In these timeslots, signals are either received or transmitted at specific time and on a certain frequency. Timing of these events require timers. SRS will be introduced next to describe a use case for timers in physical layer signaling.

2.9 SRS (Sounding Reference Signal)

SRS is an uplink reference signal that is transmitted by UE to the base station. SRS is used to measure the channel quality on uplink. More specifically, it allows to check which partial section of the full bandwidth available for certain UE has the best quality possible at that time. (3.)

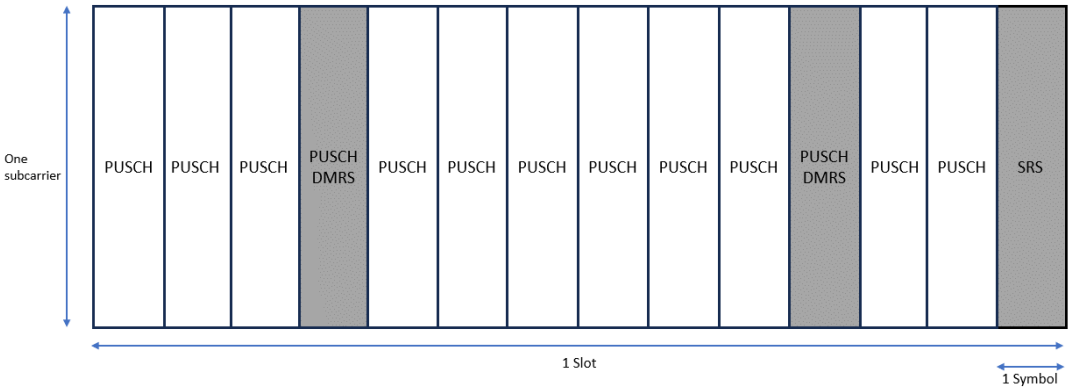


FIGURE 12. Uplink on one subcarrier during one slot duration with SRS assigned.

Channel estimation can also be done with PUSCH DMRS, but because PUSCH DMRS is transmitted only when PUSCH is scheduled and only on that bandwidth, it is not a reliable option to measure channel quality. SRS can be configured outside of PUSCH. SRS is usually configured for the full bandwidth to allow gNB to perform estimation through the whole channel band, which provides more reliable quality measurement. On a symbol level SRS is usually at the end of a slot

(Figure 13). In 5G NR SRS is more important because in TDD gNB can use the channel estimation done by SRS also in downlink. (3.) SRS is a periodical signal. It can be assigned to transmit from every two subframes to every 32 subframes. When there is multiple UE: s and the SRS overlap on multiple subcarriers UE: s can be configured in a hopping mode to have their own schedules to remove overlapping between them. (3.)

3 OPEN EVENT MACHINE

Open event machine is an open-source event driven multicore optimized framework developed for the networking data stream processing. It enables programming of scalable and dynamically load balanced multicore applications. Events, queues, and execution objects (EO) along with the scheduler and the dispatcher are the main building blocks of the EM concept. (21.)

3.1 Event machine principles

The functionality of event machine follows run to completion principle. Event machine receives an event. Events are allocated into queues. Scheduler is responsible for dequeuing events from the queue and then passing them on to event machine dispatcher according to priority. The dispatcher will call EO: s receive function when it holds an event for it. Then implementation of the EO is run. When the event has been handled and the EO:s receive function returns, the dispatcher on that core requests a new event from the scheduler and the process continues. (21.)

3.2 Terminology

Event

Event is an application specific piece of data. It describes work to be done. For example, an event can be a message or a network packet. All processing in the event machine is triggered by an event. Event type allows applications to interpret the structure of an event. Event type consists of a major and minor part. The major part specifies the type of the event for example: packet, vector or a sw buffer while the minor part is user specified and is used to distinguish between different use cases of the event. In the event machine packet flow events are sent to application specific queues. (21.)

Queue

Queues are the communication mechanism of the event machine. Events are sent to queues. Each queue belongs to some execution object (EO), but a single EO can have multiple queues. The sender chooses a queue based on what service it needs, the sender itself does not need to know which EO will process the event. (21).

Events on a queue are scheduled and processed according to following scheduling properties: name, *em_queue_prio_t* (priority), *em_queue_type_t* (queue type), *em_queue_group_t* (queue group) and an optional property *em_queue_conf_t* (configuration) that allows passing parameters to specify extra requirements if needed. (21.)

```
em_queue_t em_queue_create ( const char *                name,
                             em_queue_type_t             type,
                             em_queue_prio_t             prio,
                             em_queue_group_t            group,
                             const em_queue_conf_t *     conf
                           )
```

FIGURE 14. Create queue function and its parameters (21).

There are six queue types. The four commonly used are atomic, parallel, ordered, and unscheduled queue types. Atomic queue limits event scheduling to only one event at a time. Parallel queue type has no restrictions for scheduling, so multiple events can be processed concurrently. Ordered queue has got no scheduling restrictions either. The difference is that events coming from ordered queue will remain in the original order, even if they are processed by cores in different order. Un-scheduled queues are not connected to any scheduler. Instead the application needs to dequeue the events from the queue by using function *em_queue_dequeue()*. (21.)

Queue group defines the set of cores that will be able to process the events from the queues in the group. Queue groups can be used to separate application layers, control core deployment and load balance. Queue groups can also overlap with each other (21).

Execution object

Event machine application is built from Execution objects (EO), which are interconnected with queues. An EO consists of user defined logical functions and context data (21). The key elements of EO are receive(), start(), and stop() functions. EO is run only when EM dispatcher calls its receive() function. EM dispatcher will call EO: s receive() function when the queue holds an event for it. EO: s receive() function initializes the main part of the application's logic. On multicore systems multiple events either from the same or different queue may be dequeued in parallel or concurrently on several different cores (21). An EO may have multiple queues.

Scheduler

Scheduler dequeues events from queues. It is a SoC level function that is implemented in HW or SW. Scheduler evaluates the state of all the EM queues and gives the highest priority event to the requesting dispatcher. Scheduler gets its scheduling properties from the create queue function (Figure 14). (21.)

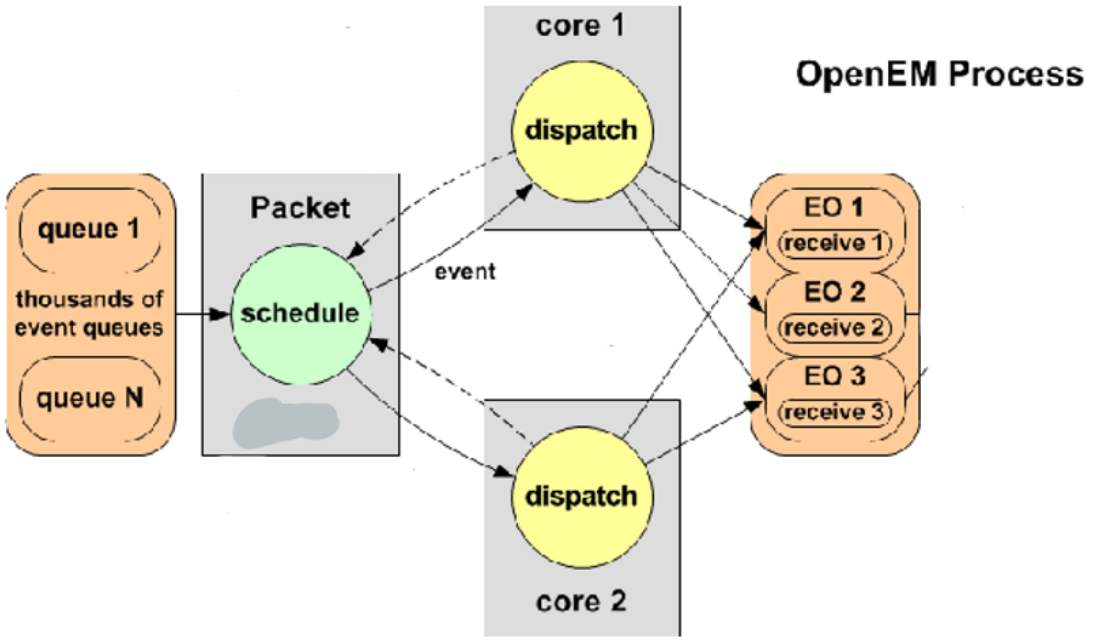


FIGURE 15. Scheduler hands events over to the dispatcher according to its scheduling parameters (16).

Dispatcher

Each core runs a dispatcher in an OS process or bare metal depending on the setup. Dispatcher requests events from the scheduler and is responsible for delivering the event to the correct EO along with the information about which queue the event or events originated from (Figure 15) (21).

3.3 Why it should be used?

Because of open event machines event-driven design, it is well suited for handling high volumes of asynchronous events while maintaining minimal latency, which is essential for real-time services or applications for example timers. Focusing on events instead of processes or threads minimizes context switching and overhead in O-EM which leads to more efficient resource usage (21).

Open event machine is designed to distribute events between processing units so that none of them becomes a bottleneck (21). This ensures optimal utilization of available resources to maintain high performance.

It is highly scalable. When correctly designed, its performance scalability is nearly linear with the number of used cores (21). In data stream processing, open event machine is better at addressing high performance, load balancing and scalability on multicore devices than traditional OS or thread-based models.

4 REFACTORING

"You start digging in the code. Soon you discover new opportunities for change, and you dig deeper. The more you dig, the more stuff you turn up...and the more changes you make. Eventually you dig yourself into a hole you can't get out of" (11, p. 6).

Refactoring is a process where every change, even a tiny one, can affect multiple things. Principles and guidelines of refactoring will be presented here and implemented as a way of working during this project.

4.1 What is refactoring?

Refactoring is a process of improving the structure of an existing program source code in a way that it does not change the external behavior of the program. It is an organized way to clean up the internal structure and design of the code that minimizes the chances of introducing bugs. Refactoring should be about applying a series of small changes, since changing a lot at once can make it too complicated and uncontrollable. During refactoring, the design of the code is improved after it has been written. (11.)

Refactoring aims to improve the design of the code. With better design the code base is easier to maintain and making changes in the future will be easier and less complex. Lastly, it makes bug fixing and other programming work more productive because the code requires less time to learn. (11.)

4.2 Principles

Testing

Having a comprehensive set of tests is essential in refactoring. The key is having self-testing code. Running the tests of self-testing code is easy and as fast to run as compiling. The amount of time spent debugging will be significantly reduced, as awareness of when and where the bug appears is always maintained. (11.)

Readability

It is important to make sure that the code is as readable as possible. Naming the functions, methods, and variables according to their use cases improves readability. It is important to follow the same naming formula throughout the code. Clarity should not be sacrificed for efficiency. Having two lines of clearly readable code is better than one line that is complicated to understand. (11.)

When not to refactor

Refactoring is a powerful tool to improve productivity, but there are cases where it should not be used. For example, large databases or class interfaces that are used outside the code base it is defined in. A good baseline would be not to refactor when the amount of work it would take is equal to the amount it would take to rewrite the code (11.)

5 REFACTORING WORK

The goal of having one timer class and EO to provide timers and timeouts for all applications that require them is aimed to be achieved through refactoring. This refactoring seeks to enhance the readability of the source code, remove duplications, and make future maintenance easier.

5.1 Planned improvements

The amount of individual EO: s is going to be reduced to one that will handle all the timer events. All similar or duplicate timer related implementations are going to be refactored into one if possible.

CMake build rules are improved where needed. CMake libraries relevant to this project are going to be refactored to contain sources and headers that exist only in that part of project tree and named accordingly. Then these libraries are linked as a dependency to where they are needed to compile an application using the timer component, instead of just copying the source file paths across the project. When CMake libraries and their configuration files are clearly named and exist in the correct locations changing and modifying of them will be a lot easier in the future.

5.2 Work process

The work began with the evaluation of which timers could be refactored and what their refactoring would require. LegacyTimerEo and TimeoutEo both inheriting BaseEo, were chosen to be within the scope of this project. They share similar class functions with each other (Figure 16). LegacyTimerEo includes a StartTimer() function to create a timer, while TimeoutEo generates only a timeout and EO, essentially duplicating functionality present in the LegacyTimerEo class.

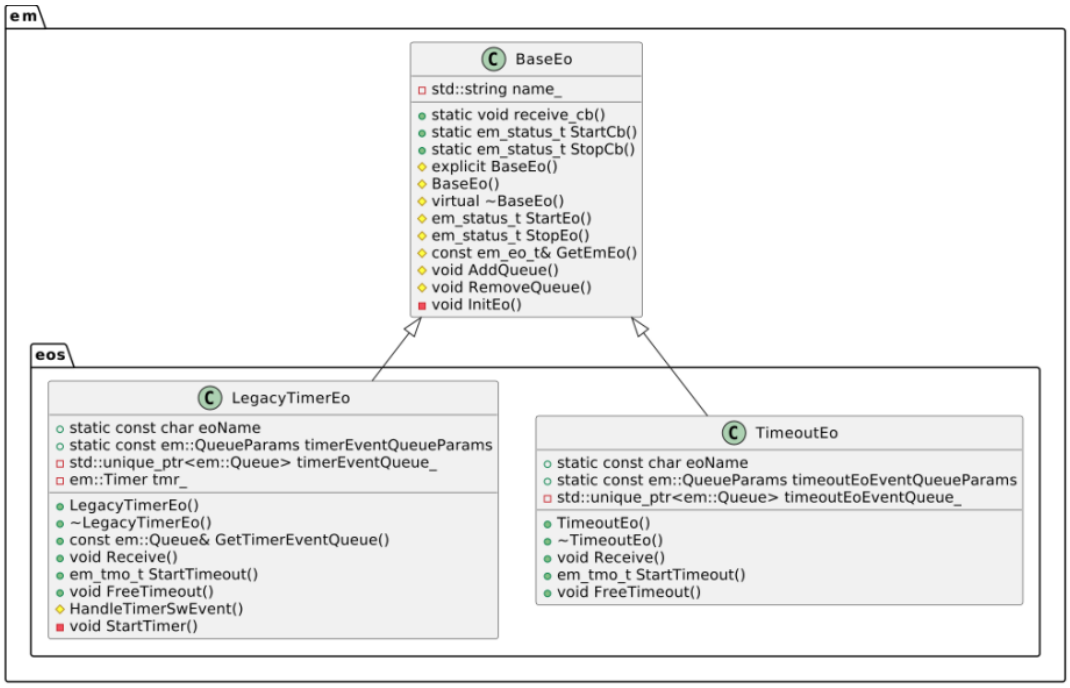


FIGURE 16. Implementation before refactoring.

The study was continued by examining how these timers were implemented and which components they were used by. When dealing with a large code base refactoring work had to be divided into small pieces. At this scale extensive changes can quickly lead to unexpected behavior. Comparison of the two classes was started. Most notable duplications were found in the EO creation and timeout creation.

The creation of the second EO was removed. Then, the timeout and timer related functions were refactored to exist in only one class. The building rules and dependencies were changed according to the new configuration, and the project was compiled, ensuring that no bugs or CMake errors were present and that no obvious mistakes were made. At this point, unit tests were not run because the unit test configuration had to be refactored first.

Refactoring of unit tests consisted of removing now unnecessary tests for the LegacyTimerEo class functions which were removed and modifying remaining tests to fit the current implementation. To do this modification of unit tests, mock tests and tests of certain subcomponents was required. When unit tests passed the refactoring commit had to go through pipeline which contains tests for unit test coverage, proper format, coding guidelines, and more extensive SCT tests which will be run in hardware. After passing those tests the commit was ready for review.

Even though the first commit was intended to be a small change, it grew larger than expected, due to the modifications required for unit tests and mock tests. Additionally, changes and improvements were made to CMake libraries regarding compilation rules, which was not completely necessary for the first commit. Retrospectively, this should have been thought out better. At this scale the change was difficult to review, and debugging was challenging.

After refactoring, implementation consists of one class which provide timer and timeouts in one EO (Figure 17). Functionality of the component remains the same.

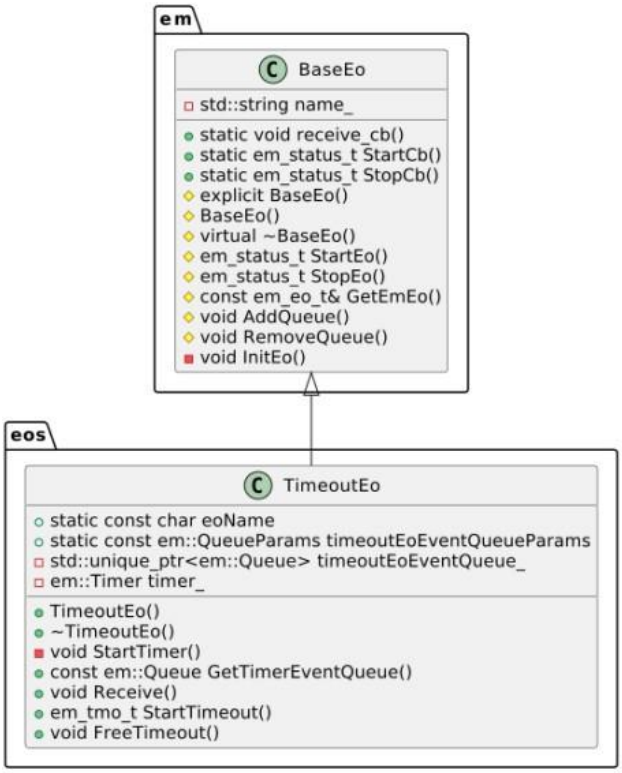


FIGURE 17. After refactoring.

5.3 Testing

Testing in this project is implemented as follows. Unit tests for all configurations are ran in google test framework. Coverage checker ensures that all program code that is executed will be tested. More extensive testing will be done within SCT (System Component Testing) framework. It injects the source code into a real hardware and performs tests on it.

Google Test

Google test is a library utilized in unit testing in C++ language. It is based on *xUnit* testing framework. Google test isolates tests by running each of them in a different object. This makes debugging easier because the test results are not dependent on each other. Also, if a single test fails, it can be run in isolation to debug it quicker. (18.)

In google test, fixtures can be used when performing multiple tests. Basic syntax includes *SetUp()*, *TearDown()* and test fixtures *TEST_F()* where the test cases are run. *SetUp()* function is used to initialize any resources or objects needed for testing. This function is always called before any test case in a test fixture, it ensures that every fixture is set up the same way before testing. Inside a test fixture, multiple test cases can be set up to test for example function output values through various assertions. *TearDown()* function is used to perform actions after the execution of tests it is typically used for cleaning up any resources allocated in the *SetUp()* function. (18.)

Google test provides support for mocking, where mock object implements the same interface as regular objects, ensuring that they behave identically. Mocking enables testing code without relying on the real implementation of external dependencies, such as APIs or services, by simulating them. By using simulated mock implementations, testing can focus on the functionality of the code itself, without interference from external APIs. (18.) For instance, a class that interacts with a database API. By mocking the database interactions, tests can concentrate on validating the behavior of the class functions without needing to connect to an actual database. This isolation simplifies testing and makes it more reliable and efficient.

SCT (System Component Testing)

SCT tests are developed to verify that a new software component integrates properly to its specified environment. SCT testing allows to test how a new feature interacts with its other internal sub components. SCT tests are run in in hardware with the new feature on top (Figure 18).

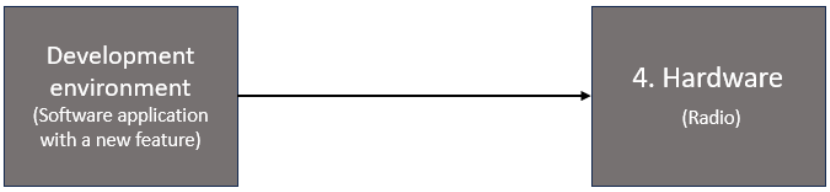


FIGURE 18. New feature is tested in hardware.

SCT will test system configurations in HW for example creating HW configurations and checking that correct responses are received from the system under testing. SCT tests will test all operations that have real-time requirements for example timers and messages having time critical information. Basic test case set in SCT consists of configuration specific setup, variations of uplink and downlink communication tests, and performance related tests. These tests verify that new software components meet the specification requirements and integrates with other software components in the configuration.

SCT tests are implemented in robot framework (Robot FW). Robot FW is a system and application independent test automation framework for test-driven development and acceptance testing (19). Robot FW is used to load needed files, services, and libraries to perform the SCT test cases.

Test cases in robot FW are written in tabular format where they are written using keywords. Test cases can be developed in any text editor or in Robot Integrated Development Environment (RIDE). Robot framework is built on Python and it runs on Jython and IronPython. Robot FW supports variables and external libraries. Test cases can also be tagged to run or skip tests. Robot FW provides reports from tests executed. Reports contain details regarding the verdict and execution time of a test. Robot FW lacks support for nested loops and conditional statements. (22.)

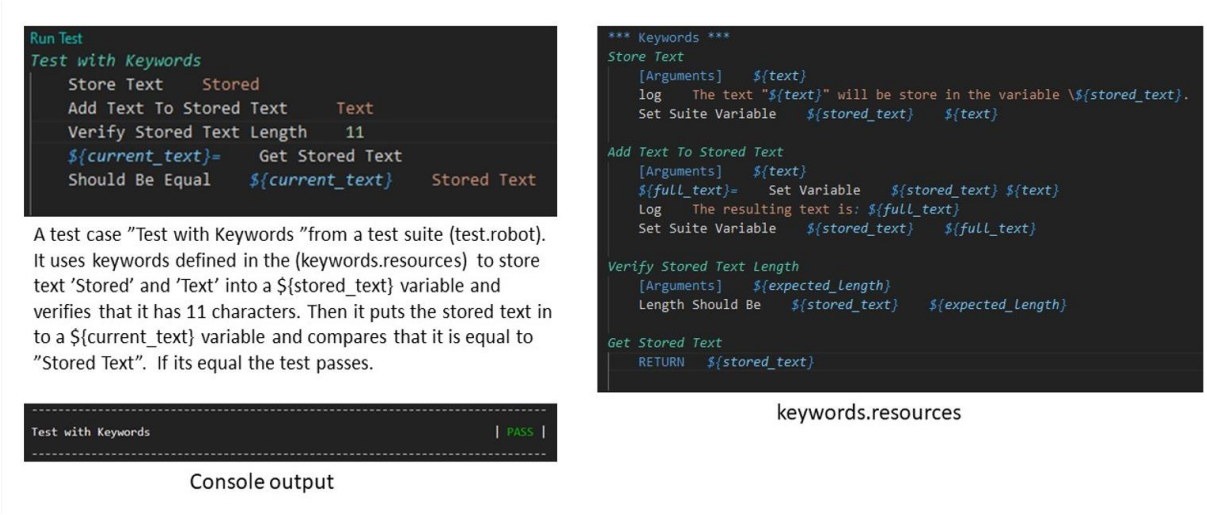


FIGURE 19. Simple example of a robot test case (19).

With robot FW, it is also possible to execute tests or scripts written in other languages for example C++ or python. In this case, the robot FW is useful in organizing and defining when and where the tests will be executed for example by tagging.

5.4 Summary

The refactoring process involved removing redundant components, consolidating timer-related functions into a single class, and updating CMake build rules and dependencies to reflect the new structure. Unit tests were modified accordingly, with unnecessary tests for LegacyTimerEo removed and existing tests modified to fit the updated implementation.

Continuous testing had an essential role throughout the refactoring process in detecting issues early and preventing the need of backtracking. Emphasizing focus on the readability of the code while refactoring made a significant impact on the outcome. By having a more descriptive implementation with reduced duplications, clarity within the codebase was enhanced, rendering it more accessible for comprehension.

While the other principles of refactoring were upheld, there was a deviation from the recommended practice of making a series of small changes. This refactoring involving the removal of nearly thousand lines of code and modification of around five hundred existing lines in a single commit, posed challenges during testing and review. Breaking down the change into smaller pieces could have expedited the process from both developer and reviewer perspective. With better planning this may

have been avoided. However, on a greater scale this principle was followed since the scope of this project was limited to these two timer components. This refactoring set the stage for possible future enhancements and refactoring work.

6 CONCLUSION

The primary objective of this thesis work was to enhance the functionality and efficiency of timer components. Through refactoring, this work successfully streamlined and optimized the codebase, eliminating redundant functionalities and consolidating necessary features into a singular implementation. As a result, the performance, maintainability, and testability of the timer components were improved.

I learnt about 5G architecture, open event machine framework, as well as had insights into the intricacies of maintaining and testing large codebases throughout the course of this project.

While this thesis focused on specific timer implementations in a real time application, there is potential for further improvement through the refactoring of other timer implementations. By establishing a framework for improvement, this study sets the stage for extending the enhancements achieved here to other timer implementations beyond the scope of this study.

REFERENCES

1. Nokia 2024. Our locations. Search date 12.05.2024. https://www.nokia.com/about-us/careers/our-locations/finland/

2. Sultan, Alain 2022. 5G System Overview. Search date 20.02.2024. 5G System Overview (3gpp.org)

3. Ryu, Jaeku. 2023. 5G/NR ShareTechnote. Search date 28.02.2024. 5G | ShareTechnote

4. Harri, Holma, Antti, Toskala & Takehiro, Nakamura 2020. 5G Technology: 3GPP New Radio. United States of America: JohnWiley & Sons Ltd. Search date 19.02.2024. O'Reilly eBooks. Access required.

5. Techplayon 2023. 5G NR-PDCCH Channel Overview and Processing. Search date 21.02.2023. 5G NR-PDCCH Channel Overview and Processing - Techplayon

6. Telecom Trainer 2023. 5G RAN Architecture. Search date 20.02.2024. https://www.telecomtrainer.com/5g-ran-architecture/

7. Cavazos, Jessy 2020. 5G NR Protocol Structure Changes – An overview. Search date 09.04.2024. https://www.keysight.com/blogs/en/inds/2020/07/23/5g-nr-protocol-structure-changes-an-overview

8. Telecom trainer 2024. 5G Protocol Stack training. Search date 08.04.2024. 5G Protocol Stack Training (telecomtrainer.com)

9. Siddiqui, Maraj, Qamar, Faizan, Ali Kazmi, Syed, Hassan, Rosilah, Arfeen, Asad & Nguyen, Quang 2020. A study on Multi-antenna and Pertinent Technologies with AI/ML Approaches for B5G/6G Networks. Search date 04.03.2024. https://www.researchgate.net/publication/366793130_A_Study_on_Multi-Antenna_and_Pertinent_Technologies_with_AIML_Approaches_for_B5G6G_Networks

10. Embedded inventor 2019. Microcontroller timers, their types, and applications. Search date 06.04.2024. https://embeddedinventor.com/embedded-timers-their-types-and-applications/#Timer_operation_modes

11. Fowler, Martin, Beck, Kent, Brant, John, Opdyke, William & Roberts, Don 1999. Refactoring: Improving the Design of Existing Code. United States of America: Addison Wesley Longman, Inc. Search date 18.01.2024. O'Reilly eBooks. Access required.

12. RF Wireless World 2018. 5G NR Initial Access Procedure | 5G NR Random Access Procedure. Search date 09.04.2024. 5G NR Initial Access Procedure | 5G NR Random Access Procedure (rfwireless-world.com)

13. Nokia 2024. Update: Open RAN explained. Search date 09.04.2024. https://www.nokia.com/networks/radio-access-networks/open-ran/open-ran-explained/

14. Jordan, Eugina 2020. The Ultimate Guide to Open RAN: Deep Dive Into RU, DU, CU. Search date 09.04.2024. https://www.thefastmode.com/expert-opinion/17962-the-ultimate-guide-to-open-ran-deep-dive-into-ru-du-cu

15. Jones, Dan, Bernstein, Corinne 2021. Radio access network (RAN). Search date 08.04.2024. https://www.techtarget.com/searchnetworking/definition/radio-access-network-RAN

16. Semantic Scholar 2014. Open event machine: A multi-core run-time designed for performance. Search date 18.01.2024. https://www.semanticscholar.org/paper/Open-event-machine%3A-A-multi-core-run-time-designed-Moerman/c2f1e59ec65a9f999a6ff90b4a331090459c90f1

17. Reis, Kyle 2024. Modulation Schemes, Coding Rates, and 4G/5G Data Speeds. Search date 10.04.2024. https://www.waveform.com/a/b/guides/modulation-coding-speeds

18. Google 2024. GoogleTest User's Guide. Search date 29.04.2024. http://google.github.io/googletest/

19. Robot Framework org 2024. Getting started. Search date 02.05.2024. https://robotframework.org/

20. Kalapatapu, Praveen & Dakshinamurthy, Sumanth 2024. Private 5G network deployments. Search date 08.04.2024. Private 5G network deployments (Enterprise Owned) | Infosys

21. Nokia Networks 2017. Event machine on ODP documentation. Search date 17.01.2024. EM-ODP: Main Page (openeventmachine.github.io)

22. Tutorialspoint 2024. Robot framework tutorial. Search date 02.05.2024. https://www.tutorialspoint.com/robot_framework/index.htm

23. Wikipedia 2024. Quadrature amplitude modulation. Search date 02.05.2024. https://en.wikipedia.org/wiki/Quadrature_amplitude_modulation

24. Mini-Circuits 2023. A Primer on Quadrature Amplitude Modulation (QAM). Search date 10.05.2024. A Primer on Quadrature Amplitude Modulation (QAM) - Mini-Circuits Blog (minicircuits.com)