Design Optimization of Safe and Secure Real-Time Systems

Jakob Menander

Kongens Lyngby 2014
IMM-M.Sc.-2014-xxxx

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk IMM-M.Sc.-2014-xxxx

Summary (English)

Many languages uses a single expression to cover the two English terms: Safety and Security. In Danish the term "sikkerhed" is used, in German they use the term "sicherheit" and even in Chinese they use only one single expression [LRNT06].

The meaning behind the term "safety", is to make sure that people and the environment are protected from harm caused by a faulty system, e.g. to protect the driver of a vehicle by releasing the airbags at impact or to prevent the impact altogether by making sure that the breaks and ABS are working as they are meant to. The definition of the term "security" is to protect information in a given system from being leaked, manipulated or forged by third parties or systems. For example we expect protection of our private information so it will not fall into the wrong hands. So one might think of safety and security as two nearly identical words which has a lot of similarities, but their objectives for protection are each other's opposites.

Conventionally, safety systems have not been concerned with security, e.g. the pressure in a steam engine secured by a safety valve, and the systems involved contained no information that could be revealed. Security related system did not have the need for using safety abilities, since security was something one would handle with a vault.

As time went by, electric and computer controlled systems, such as automatic factory machines, saw the day, but focus was mostly still on safety and not on security. With the increased use of the internet, security has become a larger part of the online universe. The internet are used to transport many sensitive

information and they are now available on more media and devices, e.g laptops, smartphones, etc. In other words, the internet allows us to communicate on different devices and exchange information. Safety systems might communicate with other systems through the internet or wireless protocols. This fusion of safety and security has made it necessary for industries that only had to think of incorporating safety in their design, now also have to incorporate security.

The aim of this thesis is to shed light on the issue of incorporating security in a safety system. Based on an existing safety system, I will come with a realistic estimate on how it can expand and also cover the fundamental capabilities in security.

I will base my work on a system called "Multiple Independent Levels of Security (and Safety)" (MILS), which is already designed to keep the integrity of the information, which is a capability in both safety and security.

Thus, security is already incorporated in the system in terms of protecting the integrity, but security also has another property, which in many systems will be described as the primarily property: confidentiality.

"Confidentiality" can be divided into two areas: Preventing information from being passed on to unauthorized persons or systems, and preventing comprehension of information if it should fall into the wrong hands. The first area creates a challenge because information should not flow downwards to a lower security level. This is exactly opposite of the integrity property in safety, where information flow is not allowed to move up a level. The second area needs to prevent a person to get valuable knowledge, if he/she should forcefully gain access to the information. This means that information has to be encrypted.

Both areas of security will be covered in the report and a proposal of how it can be implemented and which consequences a design choice will have on a system.

Summary (Danish)

Begrebet "sikkerhed" kan traditionelt tolkes på to forskellige måder. Begrebet "safety" dækker over det at sikre personer eller omgivelserne mod at tage skade fra et givet system. Det kunne være at beskytte føreren af en bil, ved at udløse airbags ved en ulykke, men det kunne også være at sikre at ulykken ikke vil indtræde i første omgang, ved at sikre at eksempelvis bremser og ABS virker efter hensigten. Omvendt dækker begrebet "security" over det at beskytte informationer i et givet system fra at blive afsløret, manipuleret eller forfalsket af personer eller systemer udefra. Eksempelvis vil vi gerne have vores private oplysninger ikke falder i forkerte hænder. Så selvom man i første omgang tænker på safety og security som to næsten identiske begreber som rummer mange ligheder, er deres mål for beskyttelse modpoler til hinanden.

Traditionelt set har safety systemer ikke haft brug for security. Damptryk sikrede man med en sikkerhedsventil og systemerne indeholdte ingen information som kunne afsløres. Security relaterede systemer havde heller ikke brug for safety egenskaber, da security ofte var noget man ordnede med en bankboks.

Sener fik man elektriske og computer styrede systemer, som automatiserede fabriksmaskiner, men fokus var stadigvæk på safety og ikke på security.

Med internettets fremhersken er security blevet en større og større del af det online univers. Internettet bruges til at kommunikere et utal af følsomme informationer og informationerne er tilgængelige på flere og flere medier og enheder. Med andre ord, internettet tillader forskellige enheder at kommunikere sammen og udveksle informationer.

Også safety systemer gør brug af at kommunikerer med andre systemer over lokale netværk, trådløse protokoller eller internettet. Denne sammensmeltning at safety og security, er med til at en påkræve at en industri som før kun skulle indtænke safety i deres design også skal til at indtænke security.

Målet med dette speciale er at belyse problemstillingen med at inkorporerer security i et safety system. Med udgangspunkt i et eksisterende safety system, vil jeg komme med et reelt bud på hvordan det kan udvides til også at dække de basale egenskaber i security.

Jeg tager udgangspunkt i et system som hedder "Multiple Independent Levels of Security (and Safety)" (MILS) og som allerede er designet til at varetage integriteten af informationer, hvilket er en egenskab i både safety og security. Dermed er security allerede inkorporeret i systemet i form af beskyttelse af integriteten, men security indeholder i midlertidig også en anden egenskab, som i mange systemer vil blive betegnet som den primære egenskab, nemlig fortrolighed.

"Fortrolighed" kan deles op i to grene: At forhindre informationer i at blive videregivet til uautoriseret personer eller systemer og at forhindre forståligheden af information hvis de alligevel skulle falde i de forkerte hænder. Den første egenskab byder på en udfordring, da det reelt betyder at informationer ikke må flyde nedad til et lavere sikkerheds niveau. Dette er stik modsat integritets egenskaben i safety, hvor informationer ikke må bevæge sig op i niveau. Den anden egenskab skal forhindre en person i at få nyttig viden, hvis han selv fremtvinger sig adgang til informationerne. Dette betyder at informationerne skal krypteres.

Begge aspekter af security belyses i rapporten og jeg giver et bud på hvordan det kan implementeres og hvilke konsekvenser et given design valg kan få for systemet.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring a M.Sc. in Informatics.

The thesis is a research project and deals with the problem of combining safety and security in embedded systems. Based on a known safety system with a layer of security integrity, I propose methods to add security confidentiality to the system.
The thesis consists of five chapters, including an introduction and conclusion, and two appendixes with abbreviations and notations.

The work has been supervised by Associate Professor Paul Pop and co-supervised by Associate Professor Christian D. Jensen.

Lyngby, 11-July-2014

Jakob Menander

Acknowledgements

I would like to thank my supervisor Associate Professor Paul Pop a lot for giving me the chance to work on this thesis. The thesis was customised to a desire to work with security in the safety domain and I thank Paul Pop for creating this thesis for me.
I would also like to thank him for his guidance, input and feedback through the work and for suggesting relevant articles.

A big thank shall also be given to my co-supervisor Associate Professor Christian D. Jensen for guidance and input of the security part of the thesis and to PhD Domitian Tamas-Selicean for review and feedback on the report.

The biggest thanks goes to my girlfriend Caroline and our daughter Ella, whom has grown from 5 to 10 months during this thesis. Without their love and support, the thesis would not have been a possibility.

Jakob Menander
July 2014, Copenhagen

Contents

Chapter 1

Introduction

The terms Safety and Security [LRNT06] have a lot in common, but they are also each other's opposites and obstructing each other. The close relation can be experienced in a lot of languages where safety and security is described together in one single word, e.g. in Danish the word "sikkerhed" covers both safety and security. In common they describe a "system" in an environment. Distinct from each other, safety aims for protecting the environment from the system, while security aims for protecting the system from the environment. Software designed with a safety purpose focuses on handling random (and maybe some periodic) faults caused by the system [HH09], while software designed for security focuses on protecting the information in the system from malicious parties.

Traditionally, embedded systems are used in the safety industry without any security, as the embedded systems operate in a closed environment, where direct access has to be obtain by an intruder to compromise the system. Systems built with focus on security are often associated with online systems with no relation to safety. But in these days where embedded safety systems are growing in size and complexity, with communication over open networks and possibly connected with the internet, security is needed to ensure the safety mechanisms.

1.1 Safety and Security Properties

Safety has two major properties; availability and integrity. In some systems, e.g. avionics, availability is more important than integrity, as the aircraft would otherwise crash. In other systems, e.g. medical instruments where the wrong dosage may be lethal, integrity is of higher importance. Despite of the importance of availability in some systems, availability is out of the scope of this thesis and I will only focus on integrity.

Security is mostly associated with its confidentiality property, but integrity is also a property in security. I will cover confidentiality as well as the integrity. It is worth to note that safety does not hold a confidentiality property, as disclosure of information would not affect the safety in a pure safety system. There are two major ways to prevent information to be disclosed. One is to control the information flow, such that a trusted message carrying secret information would never end at an untrusted endpoint. The other procedure is to prevent an untrusted intruder from getting information from a snooped message, i.e. cryptographic algorithms would prevent revealing of secret information.

Integrity is a property of both safety and security, but the meaning of safety integrity and security integrity is not the same. *Safety integrity* is the ability of a safety function to continue to be effective in spite of partial loss of its implementation measures [LRNT06]. *Security integrity* requires that an altering of the information must not be performed by unauthorised process or subject and an authorised process or subject must not make any unauthorised altering to the information. It is also required that the information will not change due to events that happen inside or outside the system that are not meant to change the information [KF09]. We can interpret *safety integrity* as a unit that is introduced to the environment and the environment will remain in a safe state after introduction of the unit, i.e. the environment will not change after the introduction of the unit. Furthermore we can interpret *security integrity* as a unit we can add information to and even if an intruder should try to change the information it can never be changed and will remain unchanged. So both safety and security wants to protect alternation (environment/information) after introducing an event (the unit/an intruder). I will in the thesis interpret *safety integrity* and *security integrity* as one single property: *integrity*, as violation of the integrity will affect both safety and security.

1.2 Security Models

As mentioned in 1.1, security is mostly associated with its confidentiality property and confidentiality is often associated with cryptographic algorithms. Another part of confidentiality is to avoid information to be leaked to untrusted parties. In 1973 Bell and LaPadula [BL73] published a security model where a downward information flow was prohibited. The model is commonly known as *no read up, no write down* and formed a basic security model that ensures the confidentiality of the information flow, by disallowing a subject to read information of a higher classification and to write information to a lower classifying object and thereby declassify information to a lower and less secure level.

Bell and LaPadula's model only focuses on confidentiality without considering integrity. In 1977 Biba [Bib77] proposed a complimentary model with a reverse information flow. Biba's integrity model is commonly known as *no read down, no write up* and prevents low integrity information from being upgraded to a higher integrity level.

Biba's integrity model is the foundation for most safety models, but is in its pure form too strong and restrictive to use in practice. One of the issues with Biba's integrity model is that information would be downgraded over time, as information could only flow from one integrity level to equal or lower integrity levels. Totel [TBDP98] proposed in 1998 a model based on Biba, but where Biba's model only has the ability to downgrade information, due to the write-down policy, Totel's integrity model preserves the integrity level and can even promote information of a lower integrity level to a higher integrity level. To do that, he introduced three kind of objects, where an object is defined as an entity providing one or multiple services to a subject or another object. The three kind of objects are (1) *Single-Level Objects* (SLO)[1] with a constant integrity level. (2) *Multi-Level Objects* (MLO) with the ability to modify the integrity level to reflect the integrity level of the invoker. They have no "memory" and they restore their integrity level when freshly created. The third (3) is a *Validation Object* (VO), which has a single level of integrity, but takes input from redundant or diverse objects with a lower integrity level. The output would be at the same integrity level as the VO itself. E.g. a VO would take input from low level sensors to validate them together to a single high level output.

There arises a big problem when systems grows bigger and more complex: The certification of the system. In Totel's integrity model the whole system would be certified as once. There was a need to divide the system up in smaller pieces, easier to certify. Rushby [Rus81] introduced in 1981 the concept of separate

[1] A full list of abbreviations can be found in Appendix A.

subsystems. A *Separation Kernel* (SK) isolates processes from each other and the single subsystems could now be certified individual. This made it much easier to design and maintain more complex safety systems.

1.3 Multiple Independent Levels of Security (and Safety) (MILS)

Based on the concept of separation, the *Multiple Independent Levels of Security (and Safety)* (MILS) approach was described in 2005/2006 [AFHOT06]. MILS consists of three layers [BDRS08], with the SK as the lowest layer. Next comes a *Trusted Subsystem* (TSS) ensuring the communication between the applications. On top of that is the last layer where the untrusted application services are executed.

Where Totel's integrity model is designed to run on a single processor [WM12], the MILS is designed for *Multi-Processor System-on-a-Chip* (MPSoC) devices. To support the architecture a *Time-Triggered Network-on-a-Chip* (TTNoC) was introduced in 2010 [WESK10] as the communication network. This ensures both a spatial and temporal separation in the transportation of messages.

The *Time-Triggered* (TT) network is preferred over an *Event-Triggered* (ET) network. An ET network may deliver the message from an asynchronous event faster than TT, but the messages may also be delayed if many events are triggered at the same time. TT may not send a message at the occurrence of an event, but every process is guaranteed a sending slot in an a priori known point in time and within a given deadline. This makes TT less flexible than ET, but more deterministic, fault tolerant and manageable for the designer of the architecture [Alb04]. The deep integration of TTNoC makes MILS a *Time-Triggered Architecture* (TTA).

MILS consists of *components* (or *μComponents*) [WESK10] connected together by a TTNoC. The components can be assigned independent security levels, which will affect the possible information flow, but also affect the cost for validation in time and money. The higher the level, the higher the cost. A *Trusted Interface Subsystem* (TISS) provides the interface between the components and the TTNoC. A *Trusted Network Authority* (TNA) manages the routes in the TTNoC and a *Resource Management Authority* (RMA) guards for unauthorised changes of the TNA. The routes are called *Encapsulated Communication Channels*, which means they are unidirectional communication channels with one sender and one or several receivers at a specific point in time. To cope with the inflexibility of Biba's integrity model, a *middleware* can be placed between

the application and the TISS. This middleware can be designed to validate redundant input and functions in the same way as the VO in Totel's integrity model.

MILS is basically an integrity model with a downward information flow. Cryptographic algorithms can be added to ensure the secrecy of information and thereby adding a bit of confidentiality to the system. It is possible to reverse the information flow [WESK10] from downward to upward and thereby get a pure confidentiality model, but then the integrity is neglected.

1.4 ACROSS MPSoC

MILS is only described theoretical and the industry lacks a MPSoC system with focus on safety. An European project, the ARTEMIS ACROSS project, was formed in 2010[2] to come up with such architecture [SEH+12]. The result was the ACROSS MPSoC architecture; a MILS system. Small variations in the descriptions of the architecture can be found, e.g. the TNA and RMA described in [WESK10] are combined to a *Trusted Resource Manager* (TRM) in the descriptions of the ACROSS MPSoC architecture [WM12].

Further in this thesis I will use the ACROSS MPSoC architecture and describe it in extensive details.

1.5 Attacker model

But why is security and specially security confidentiality needed in a safety system? Of course the integrity needs to be preserved even after a malicious attack or it would endanger the safety, but what information in a safety system needs confidentiality?

An intruder may have several interests in attacking the system. He might want to drain the system for information by eavesdropping on the communication, he might want to take control of the system or simply to put it out of function. The intruder can choose to attack the components, the communication channel or a combination of both.

A car is a safety system we also want to be secure. Most cars these days are

[2]The ACROSS project was closed again in 2013. `http://www.across-project.eu`

relying more and more on embedded systems, so called *Electronic Control Units* (ECU), and in a near future most car will have steer- and brake-by-wire, i.e. mechanical and hydraulic will be substituted by ECUs. As said, most modern cars contain a lot of ECU, but the architecture binding the ECUs together offer no security. In new cars, it is possible to connect a mobile phone to manage and upgrade the GPS, provide easy handfree communication, listen to music stored on the mobile device, etc. This means that a you can connect a device that is not validated for safety nor security, to the internal system. A mobile phone has often access to the internet and an attack via the internet through the mobile phone, could give a malicious intruder access to the internal systems. One can also conceive that access to the internet integrated directly in the internal system, is not of a distance future.

But an attack on a safety system may not only be at runtime. It is reasonable to conceive that attacks take place in the development phase. A malicious employer may implement a backdoor or malicious code may find its way through the internet on the machine the system is developed on. Even the use of USB sticks can cause malicious code to be implemented into the system.

But what makes a malicious attack on a car desirable for an intruder? By eavesdropping, an intruder can listen to conversations in the car, as more and more cars have a microphone integrated to enable the driver to talk handless in his mobile phone. The information (e.g. the contact list or messages) on a mobile phone connected to the cars integrated system can also be leaked. Information on the cars position (GPS) or its general status (e.g. the speed or odometer) can also be of interest of an eavesdropper.

If messages from the ECUs is kept in a black box for future investigations, e.g. after a traffic accident, altering messages can be used in assurance fraud. Altering messages can also be used to make a car appear less used by altering the data of the odometer, which will result in a higher sales price if it is resold.

Deleting messages to the inflater of an airbag, would cause the airbag to not function. In a combination with deleting messages from the foot brake sensors the result could be fatal, as the car could crash without inflating the airbags. It could also just be limited to an annoying character, by disable e.g. the heater, air condition, windows or even the engine.

Adding information could result in executing commands and more or less taking over the system. Annoying functions could be activated (e.g. activating the horn in a car), but also potential dangerous functions as releasing the airbags at full speed or turn off the light of a car, can be activated. In cars with drive-by-wire, an intruder can also take control by steering the car.

With this in mind, tomorrow's safety systems cannot rely on safety alone any more. Security need to be added to ensure the safety properties.

1.6 Contribution

The ACROSS MPSoC architecture is a safety architecture with a layer of security integrity and described in 2012 [WM12]. It is based on the MILS architecture from 2005/2006 [AFHOT06]. The architecture I use is therefore described earlier and is not new knowledge. My contribution is to analyse and suggest a method to add confidentiality to the ACROSS MPSoC architecture. A similar approach has been described in [WESK10], where a confidentiality flow is described at the expense of the integrity flow. In my system the integrity flow is preserved along with a confidentiality flow.

The information flow is only one part of confidentiality; the other part is the secrecy. I bring a Secure Channel [IW13] into the system to provide end-to-end encryptions for communication over external network. For long lasting encryption I have proposed the use of a special Crypto Component and analysed what has to be take into account.

Chapter 2

Application Model

Integrity has been in focus in most MPSoC models. To extend such a model to also focus on confidentiality, we must provide some mechanisms to enforce confidentiality without compromising integrity. These mechanisms are primarily in the architecture, but in order understand and improve the architecture, we must understand the application model as well.

The proposed application model is an adoption and slightly reformulated version of the application model in [TSP13]. An entire list of notations can be found in Appendix B.

2.1 Notation

An application $\mathcal{A}_i$ is a direct, polar and acyclic graph $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)$ and the set of all applications is specified as Γ. The application graph $\mathcal{G}_i$ consist of all nodes $\mathcal{V}_i$ and all the edges between the nodes $\mathcal{E}_i$ in the given application subsystem $\mathcal{A}_i$. Each node represent one task $\tau_j \in \mathcal{V}_i$. All nodes are mapped to *Processing Elements* (PE) by the function $M : \mathcal{V}_i \to \mathcal{N}$ and a task in the node is associated to exactly one partition slice $\phi(\tau_j) \to p_{ij}^k$ where $\tau_j \in \mathcal{V}_i, \phi : \mathcal{V} \to \mathcal{P}$.

A partition is denoted P_j and a set of partition slices on N_i is denoted $\mathcal{P}_{ij}$. The k^{th} partition slice is denoted p_{ij}^k. The scheduling of tasks to partitions is made using *Static-Cyclic Scheduling* (SCS). In some cases two tasks are not allowed in same partition, e.g. two redundant tasks may not share partition, as both tasks could be affected of a failure in the partition. A *Protection Requirement Graph* $\Pi(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of all tasks and $\mathcal{E}$ is the dependencies between them, enforce prohibition of sharing partitions. The edge $sr_{ij} \in \mathcal{E}$ means that τ_i and τ_j are not allowed in the same partition.

The edge $e_{jk} \in \mathcal{E}_i$ has output from τ_j and input in τ_k. A task must receive all its input before its ready and will first output messages after termination. Messages m_i are used by tasks, located on different PEs to communicate with each other. The size s_{m_i} of m_i are known. The deadline $D_{\mathcal{G}_i}$ has be to reached within the period $T_{\mathcal{G}_i}$ for each $\mathcal{G}_i$, i.e. $D_{\mathcal{G}_i} \leq T_{\mathcal{G}_i}$. The *Worst-Case Execution Time* (WCET) C_i are known for task τ_i. Messages can only be sent at a priori known point in time according to a time scheme. The period of the time scheme T_{cycle} is repeated continuously. The T_{cycle} is divided in several *Major Frames* (MF) with a length given by the designer and with a period denoted as T_{MF}. The partitions are grouped together in MFs.

An integrity level $IL : \mathcal{V}_i \rightarrow \{ILk\}$, where $k \in \{0, \ldots, 4\}$ (covering the integrity of both safety and security) and a confidentiality level $CL : \mathcal{V}_i \rightarrow \{CLk\}$, where $k \in \{0, \ldots, n\}$ is assigned to every task in order to determine the restriction of the information flow and certification. A task τ_i is assigned both an integrity level $IL(\tau_i)$ and a confidentiality level $CL(\tau_i)$ independent of each other.

2.2 Rules

There are some rules to follow to ensure the safety and security requirements. The rules are an adaptation of the rules proposed in [WM12], with some addition to cover not only integrity, but also confidentiality.

Rule 1: A task is placed in exactly one partition slice and is allowed to share partition with another task *iff* the two tasks do not share an edge in Π and they have same integrity and confidentiality level.

The integrity level and the confidentiality level of a task may be set so low that they will be non-critical and not impact the safety and security of the system. Even though, both an integrity level and confidentiality level must be assigned a task.

Rule 2: A task is assigned exactly one integrity level and exactly one confidentiality level.

Communication between tasks is done by passing messages over the communication channel. To apply Biba's integrity model [Bib77] we must ensure a downward information flow.

Rule 3: The information flow is allowed, only if the sending task has a higher or equal integrity level than the receiving task; $IL(\tau_{send}) \geq IL(\tau_{receive})$.

In a model dealing with both integrity and confidentiality, the upward confidentiality flow proposed by Bell and LaPadula [BL73], apply as well as the downward integrity flow. The confidentiality flow is an extension of the original rules proposed in [WM12]

Rule 4: An information flow from one task to another is allowed only if the sending task has a lower or equal confidentiality level than the receiving task; $CL(\tau_{send}) \leq CL(\tau_{receive})$.

Rule 3 enforces a rigid downward information flow, where information can only flow downward, and that is not practical to work with. The downward flow can be circumvented in rule 5 by allowing an upward flow, if the information is validated to a higher integrity level.

Rule 5: Messages from a task with a low integrity level to a task with a higher integrity level must pass through a *Validation Middleware* (VaM). The VaM must receive information from several different redundant or (even better) diverse tasks, with a lower integrity level than the VaM. The information must be received within a given time span.

While rule 5 relaxes the information integrity flow in a safe and secure manner, where information is validated and upgraded to a higher integrity level, the confidentiality flow is relaxed in a way where sensitive information is filtered out of the information flow.

Rule 6: Information can flow from a task with a high confidentiality level to a task with a lower confidentiality level *iff* sensitive data is filtered out of the messages or the message is protected by encryption with no possibility to decrypt at the receiving task.

As long as tasks are communicating on-chip through the TTNoC, eavesdropping is not possible. But off-chip communication cannot guarantee the confidentiality of the information.

Table 2.1: The three colons from left indicates a task and its configuration. The colon to the right indicate the possible flow to other tasks. τ_2 can only send messages to other tasks with the same configuration, while the configuration of τ_3 allows it to send messages to every other task without regard to their configuration.

τ_i	$IL(\tau_i)$	$CL(\tau_i)$	$\tau_i \rightarrow \mathcal{V}_j$
τ_1	L	L	$\tau_1 \rightarrow \{\tau_1, \tau_2\}$
τ_2	L	H	$\tau_2 \rightarrow \{\tau_2\}$
τ_3	H	L	$\tau_3 \rightarrow \{\tau_1, \tau_2, \tau_3, \tau_4\}$
τ_4	H	H	$\tau_4 \rightarrow \{\tau_2, \tau_4\}$

Rule 7: Information must be encrypted before sending through an external network.

2.3 Safety and Security Level

Rule 2 dictates that a task is assigned exactly one *Integrity Level* (IL), covering both the safety integrity and the security integrity, and one *Confidentiality Level* (CL). The consequence of these two separate levels is that messages can flow free from a task with a high IL and low CL, while information from a task with a low IL and high CL are limited to send messages to other components with the same configuration. A simple example where the security levels can be either *High* (H) or *Low* (L) is shown in Table 2.1. The table illustrates the allowed flow from one configuration of tasks to another configuration. The three colons to the left indicates a task τ_i and its IL and CL configuration. The colon to the right indicates the possible flow from τ_i to other tasks with different configurations. One could argue to combine the two security levels as one single level with four configurations. But in reality the security levels are not limited to just a high or low configuration. It is easy to see the increased complexity in Table 2.2, where three levels of IL and CL are used. To keep the allowed communication routes simple and comprehensible, I have therefore chosen to keep the IL and CL separated. The IL and CL do not affect each other, but they affect the overall information flow between two tasks, i.e. a restriction in the confidentiality flow will not make any restrictions to the integrity flow and vice versa.

Table 2.2: A more complex model than Table 2.1, where τ_3 can only send messages to other tasks with the same configuration, while the configuration of τ_7 allows it to send messages to every other task without regard to their configuration.

τ_i	$IL(\tau_i)$	$CL(\tau_i)$	$\tau_i \rightarrow \mathcal{V}_j$
τ_1	1	1	$\tau_1 \rightarrow \{\tau_1, \tau_2, \tau_3\}$
τ_2	1	2	$\tau_2 \rightarrow \{\tau_2, \tau_3\}$
τ_3	1	3	$\tau_3 \rightarrow \{\tau_3\}$
τ_4	2	1	$\tau_4 \rightarrow \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$
τ_5	2	2	$\tau_5 \rightarrow \{\tau_2, \tau_3, \tau_4, \tau_5\}$
τ_6	2	3	$\tau_6 \rightarrow \{\tau_3, \tau_6\}$
τ_7	3	1	$\tau_7 \rightarrow \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9\}$
τ_8	3	2	$\tau_8 \rightarrow \{\tau_2, \tau_3, \tau_5, \tau_6, \tau_8, \tau_9\}$
τ_9	3	3	$\tau_9 \rightarrow \{\tau_3, \tau_6, \tau_9\}$

Table 2.3: ISO/DIS 26262 SIL decomposition scheme [TSP13]. Shows the possible decomposition of a SIL.

SIL	Can be decomposed as
SIL 4	SIL 4 or SIL 3 + SIL 1 or SIL 2 + SIL 2
SIL 3	SIL 3 or SIL 2 + SIL 1
SIL 2	SIL 2 or SIL 1 + SIL 1
SIL 1	SIL 1

2.3.1 Safety Level

Industrial standards as the *Safety Integrity Level* (SIL) used to dictate the development process and the certification procedure of safety related functions [TSP13]. SIL are operating with four levels, with SIL 1 as the lowest level and SIL 4 as the highest level. The higher the level, the lower the tolerable hazard rate, i.e. the SIL can be associated with the tolerable hazard rate [LRNT06]. There is a SIL 0, but it is assigned to non-critical tasks and are not covered by the standards [TSP13]. I will use the notation IL instead of SIL, as SIL refer to safety and this thesis operates with both safety integrity and security integrity, i.e. IL covers both safety and security.

A high IL ensures a high level of safety, but it also cost time and money to develop high IL and get it certified. To circumvent the high cost of a task with a high IL, the task can be decomposed into two redundant tasks with lower ILs in the same way SIL is decomposed, see Table 2.3 [TSP13]. A decomposed IL would not affect the CL, i.e. the two new decomposed tasks will inherit the original CL. By decomposing a task, the number of tasks in the system increases.

This can potentially reduce the schedulability, due to the extra tasks that have to be placed in the schedule table.

An IL can always be elevated to a higher IL, if it is not at the highest level yet. A high IL costs more, but can be necessary to obtain a schedulable solution. Two tasks of different IL (or CL) cannot share a partition, and tasks with low IL may need elevation in order share a partition with task with higher IL.

2.3.2 Security Level

There are no standards for the number of levels in CL. Various levels of confidentiality can be applied to a system. A common toy-example is using three levels: *Unclassified* (UC), *Secret* (S) and *Top-Secret* (TS). I will use these three levels in the further description of CL in this paper. Comparable, but not equivalent to the concept of SIL for safety, the security has a concept known as *Evaluation Assurance Level* (EAL) [LRNT06]. The EAL is a standard for certifying security functions considering all the security properties, i.e. including integrity and confidentiality. There are seven EAL levels and they correspond to assurance levels [CC12]: (1) Functionally tested, (2) Structurally tested, (3) Methodically tested and checked, (4) Methodically designed, tested and reviewed, (5) Semiformally designed and tested, (6) Semiformally verified design and tested and (7) Formally verified design and tested.

Where a safety function can be certified *to* a SIL, the security function can be certified *at* an EAL, i.e. a particular level in EAL only tells us how much a security function has been tested and how much we can trust it to be as secure as it claims, but not how secure the function really is. Therefore I will not use the EAL as a guaranty for the security in the CL. Another difference between IL and CL is that a CL cannot be decomposed nor elevated as the IL.

2.4 Application Examples

The application model is illustrated in Figure 2.1. Here are shown two application subsystems $\mathcal{A}_1$ and $\mathcal{A}_2$ and their dependencies. Tasks communicate by sending messages along the edges. A message has exactly one sender, but can have several receivers. In (a) τ_1 sends messages to τ_2 and τ_3. The message from τ_1 can be a multicast message from one sender to two receiving tasks. It can also be two different messages sent at different time from one sender to one receiver.

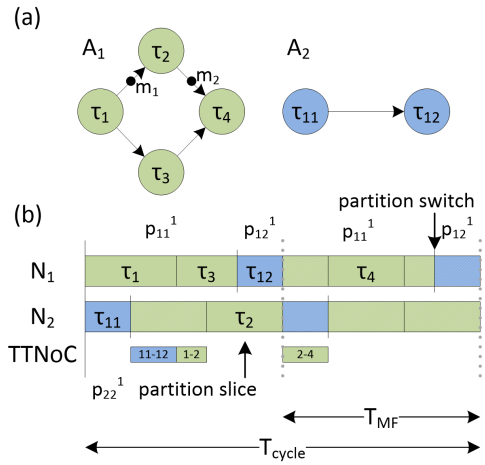

Figure 2.1: The graph in (a) shows two flows. A flow in $\mathcal{A}_1$ from τ_1 to τ_4 and a flow in $\mathcal{A}_2$ from τ_{11} to τ_{12}. (b) shows the tasks from (a) scheduled to partition slices and mapped on two PE.

τ_4 receives two individual messages at different point in time. A task that take input from other tasks cannot execute before all inputs are received.

The mapping assign one task τ_i to one partition slice p_{ij}^k. The schedule is shown in (b) with two PEs N_1 and N_2. τ_1, τ_3, τ_4 and τ_{12} are placed on N_1 while τ_2 and τ_{11} are placed on N_2. Unused partition slices are greyed out. The message from τ_1 to τ_2 and τ_3 is multicast at the same timeslot, but as τ_1 and τ_3 are placed on same PE, only the message to τ_2 is going through the TTNoC. τ_4 receives input from τ_2 and τ_3. As τ_3 shares the PE with τ_4, only the message from τ_2 is going through the TTNoC. If the three tasks was placed on three different PEs, τ_2 and τ_3 would need two different timeslots to send their messages to τ_4. τ_4 cannot start its execution before it has received all input messages.

The possible information flows outlined in Table 2.1 is illustrated as a graph in Figure 2.2. It is easy to see that the configuration of high integrity level and low confidentiality level in τ_3 has a free outgoing flow, while it cannot receive information from other configurations. In contrast, the configuration in τ_2 can only receive information, but not send to other configurations (other than equal configurations - not pictured).

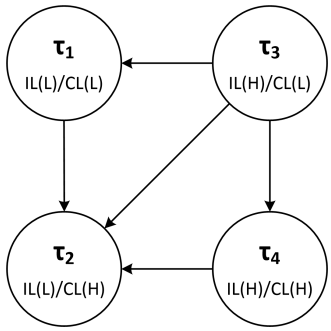

Figure 2.2: Illustration of the possible flow between tasks with different IL and CL configurations introduced in Table 2.1. It is easy to see that information from τ_3 can flow free, while information from τ_2 has a restricted flow.

Chapter 3

Architecture Model

I have chosen the ACROSS MPSoC architecture to support the application model. The ACROSS MPSoC is designed as a combined safety and security architecture built on a MPSoC platform [WM12]. The architecture consists of multiple components connected together by a TTNoC. The term "component" (or "μComponent" in some articles) is used in various articles, e.g. [AFHOT06], [BDRS08], [ESOHK08] and [WESK10], to describe the part of the architecture providing the application specific services. The component consist of a *host* and the *TISS*, as I will discuss later in Section 3.1.1 and 3.1.2. None of the articles have a deep description of the application model and I interpret the use of the term "component" in the articles, as a label to talk about components in an abstract way. If we remove the label "component" we could talk about the host communicating with other hosts through the TTNoC via the TISS. I will use the term "component" in its abstract form in this thesis and consider the host of the component to be the application task.

The ACROSS MPSoC architecture is a *Multiple Independent Levels of Security* (MILS) system [AFHOT06], [BDRS08], [WESK10], where the fundamental idea is to separate subsystems. The concept of separation was introduced by Rushby[Rus81] in 1981 and ensured by a trusted separation kernel. The architecture consists of three parts: the hardware layer with the separation kernel, a trusted part and an untrusted part. The trusted part ensures the core services and cannot be changed by the application specific services. The untrusted part

performs the application specific services and can incorporate middleware to relax the strict information flow.

The security in the ACROSS MPSoC architecture only covers the integrity, with a limited aspect of confidentiality. The architecture's limitation is that it will only support confidentiality in form of cryptographic algorithms, but does not support an upward information flow, i.e. the information flow would be downward (integrity) and thereby excludes an upward flow (confidentiality). An upward flow was described in [WESK10], but the downward flow was neglected.

3.1 The Architecture

The lowest layer in the architecture is the hardware and the *Seperation Kernel* (SK) as mentioned in section 1.3. The hardware is out of the scope of this thesis and will not be covered. The SK is the core concept in the ACROSS MPSoC architecture and isolates processes in separate partitions on a shared processor [WESK10], [Rus81]. The partitioning, which is both spatial and temporal, enforces (1) data separation, (2) the information flow by using inter-partition communication, (3) sanitisation by cleaning any shared resources and (4) damage limitation, as a fault in one partition would not affect other partitions [AFHOT06].

Each component is assigned to a partition slice and has assigned exactly one level of safety and exactly one level of security. The partition slices are scheduled on PEs using *Static-Cyclic Scheduling* (SCS). In contrast [WESK10] writes: *"Each partition is mapped to exactly one component and each component hosts at most one partition."* By doing so, a task in a component is separated from other tasks and a failure in one partition will not propagate to another partition. The information flow is also ensured, as information cannot flow outside the partition from one task to another. But the scheduling would be hard to optimise, if partitions cannot be shared and I choose to apply the tasks to partition slices instead of partitions. There are some precautions to consider. Information can flow inside a partition and tasks sharing a partition must therefore be configured with the same IL and CL. Some tasks may not share partitions and is connected through an edge in Π as described in Section 2.1.

To function it is required that the SK is always available and invoked, tamper-proof, non-bypassable and free of design faults. To ensure that, the SK must be easy to certify and is thereby kept as small and simple as possible. The SK is the *Trusted Computing Base* (TCB) of the system. Lampson [LABW92] is often quoted for describing the TCB as a *"small amount of software and hardware that*

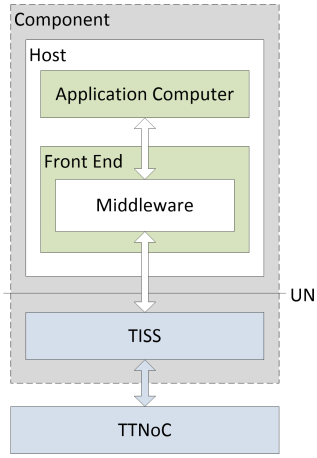

Figure 3.1: The architecture of a component. It consists of two architecture elements: the TISS which also is part of the TSS and the host in which the application specific services are placed. Via the UNI have the host a transparent interface through the TSS to other hosts.

security depends on and that we distinguish from a much larger amount that can misbehave without affecting security."

3.1.1 The Trusted Architectural Layer

The application specific services are carried out by architectural elements called *components* (or *μComponents*) as introduced in Section 3, and are connected through a *Time-Triggered Network-on-a-Chip* (TTNoC). Figure 3.1 shows the component and its elements.

The TTNoC is a part of a *Trusted Subsystem* (TSS). TSS is composed of: the TTNoC, a *Trusted Interface Subsystem* (TISS) and the *Trusted Resource Manager* (TRM). Together they form a black box for the components and is assumed to be free from design faults. All communication is carried out by the TSS, transparent to the component. By using the TTNoC as the internal communication network provides us with some fundamental security functionalities [WESK10] such as: data isolation, a controlled information flow and damage limitation. For further descriptions on how the TTNoC works, I recommend reading [Sch07].

The message routes in the TSS are called *Encapsulated Communication Chan-*

nels and are unidirectional communication channels with one sender and one or several receivers, which transport the message in a priori known point in time. The endpoints of the encapsulated communication channels are called *ports* and are located in the TISS. Ports leading out of the SoC to another SoC are called *gateways*. Due to security reasons communication through gateways is encrypted, as specified later on in Section 3.1.2.2, and gateways are therefore limited to special IO-components, i.e. ordinary components cannot contain gateways and can thereby not connect to an external network.

The routes and ports are managed by the TRM according to a *Time-Division Multiple Access* (TDMA) scheme [OH11]. Only the TRM can re-/configure the routes and ports, and acts as a guardian for reconfiguration. In earlier articles the TRM is often called *Trusted Network Authority* (TNA) and is often co-operating with a *Resource Management Authority* (RMA), where the RMA reconfigures the communication and the TNA guards the activities of the RMA [PPES09]. In ACROSS MPSoC the TNA and RMA are combined into TRM. A component cannot change the encapsulated communication channels (this is exclusive managed by the TRM), but a component can suggest a reconfiguration to the TRM. To manage and reconfigure the encapsulated communication channels, the TRM knows the TDMA, the components configuration, the components safety and security level and the configuration of the TTNoC. The TRM makes sure that no safety or security policy is violated during a reconfiguration. The TRM has the communication channels under constant surveillance, preventing unauthorised alternations. The TRM is also checking the identity of the component and allows only authorised components to communicate. With TTNoC, TRM and TISS combined in the TSS, the TSS ensures a time-triggered communication, a common time among the system and integrated resource management.

The TISS forms one part of a component, as shown in Figure 3.1, and act as a guardian, by only accepting messages to be sent or received according to the TDMA. This prevents a faulty component from being a "babbling idiot"[1]. Even though the TISS is placed in the component, the TISS can only be reconfigured by the TRM. The other part of a component is called the *host*, see Figure 3.1. Where the TISS are part of the TSS and certified as the highest level of the system, the host is part of the untrusted area and must be individual certified. The TISS provides a *Uniform Network Interface* (UNI) to the host, so when a task in the host want to communicate with other tasks it connects to the UNI and the transportation of the message(s) are transparent to the task.

[1] A babbling idiot is a faulting node, flooding the communication network and taking up resources. It can potentially prevent correct functional nodes in receiving and sending messages or making the node repeatedly executing its application service inappropriately many times.

3.1.2 The Top-Layer

The designer of tasks is restricted from altering the TSS, but has access to the host which is composed by an *Application Computer* (AC) and by *Front End* (FE). As described in the introduction to Section 3, I consider the host to be the application task. A task contains therefore both the AC and the FE. The AC performs the application specific services and the FE services as an extension to the communication services, i.e. the application specific parts of the task are performed by the AC in the architecture, while parts of the task extending the communication service are performed by the architectural FE.

Middleware services are extensions in the FE, which provide high level communication services to e.g. circumvent the unidirectional flow. The partitioning of tasks will therefore include both the application specific services and the serviced performed by the middleware in one partition slice. That also means that the WCET for a task increases by using middleware. Even though I consider the host as a task when mapping tasks to partition slices, I will refer to tasks as just the AC. The reason for this is to better explain the behaviour of the system and focus on the application specific part of the task.

A dual-ported memory denoted as *Port Memory* [PPES09] is also located in the FE. Messages from the task have to be written into the Port Memory and forwarded by the TISS onto the TTNoC at an a priori known point in time. A component is applied the same IL and CL configuration as the tasks, with the same restrictions, as described in Section 2.3.

3.1.2.1 Middleware

Middleware is used to provide an extra layer to the communication services provided by TSS and does not affect the application. The extra layer is used to relax the rigid information flow (downward for integrity and upward for confidentiality), by allowing a reverse flow, i.e. middleware allows us to create an upward integrity flow and a downward confidentiality flow.

As discussed in Section 1.2, we need to be able to upgrade information. E.g. three redundant braking sensors in a car has usually a low IL, but have great consequences if not working properly. In a safety integrity manner a sensor is "likely" to fail to output a value. In the perspective of security integrity, the value produced may not be accurate, it might even be a false measurement or could be produced by a deliberate action caused by a malicious intruder.

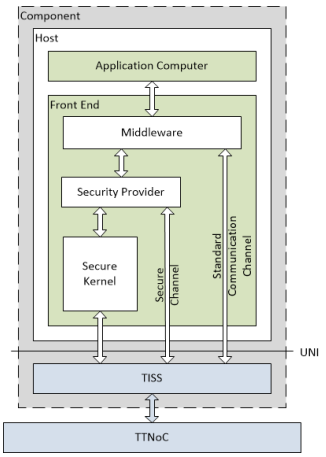

Figure 3.2: Here the SC is implemented along with a piece of middleware. Both the SC and the middelware are placed in the FE, such that SC is placed between the network and the middleware. The middelware is placed with SC and AC on each side.

To ensure a trustworthy measurement with a higher IL, we need to gather information of several redundant or diverse sources. A *Validation Middleware* (VaM) [WM12] gathers the redundant low IL input and runs a voting algorithm among the values to output a single trustworthy high IL value. The VaM is located at the receiving component and certified at the same IL as the host component.

The confidentiality flow can be circumvent by a *Flow Control Middleware* (FCM) [WESK10]. Not all information in a component with a high CL may be sensitive and can apply to a lower CL as well. As an example, only the information to identify a person on his patient journal is sensitive, while the diagnose and treatment are not of a sensitive nature and rather useless without the identification of the patient. The FCM provides a filter to remove sensitive confidential information and lets insensitive information through. In this way a downward information flow can be accepted. The FCM are located at the sending component, allowing it to send information with a lower CL than the original CL of the host component.

3.1.2.2 Secure Channel

A *Secure Channel* (SC) [IW13] is a common design pattern that ensures the secrecy in the information flow to external communication. The message is sent through the TTNoC to a special IO-component with a *gateway* to the external network and SoC. The SC ensures the secrecy if the message is eavesdropped and is placed in the FE and is a piece of middleware. If other middleware services are assigned to the component, SC is placed between the middleware and the TISS as shown in Figure 3.2.

A *Secure Kernel* manages and generates the cryptographic keys used by SC. The key generation demands heavy computation, which could be a problem in a resource limited MPSoC. For that reason it is hardware implemented to ensure faster and resource-saving computation. The Secure Kernel must ensure that a key is ready to use, at the point in time a message needs it. The Secure Kernel is not to be confused with the Separation Kernel (SK) of the ACROSS MPSoC architecture. It is not part of the TSS, but is the TCB for the SC.

A *Secure Provider* executes the encryption and decryption and provides the task (or middleware if implemented) with a transparent channel with security properties. The architecture also provides a standard communication channel to bypass the SC for on-chip communication where encryption is not needed.

The SC encrypts information going through the TTNoC, but is used for off-chip communication to ensure confidentiality on an unsecured external network. A special IO-component with gateways to the external network, e.g. *TTEthernet*, must be used for this case. Then messages must travel through the TTNoC, to the IO-component, further to the external network and arrive at the destination SoC. More detail description is given in description of the behaviour in Section 3.5.2.

The cryptographic algorithm used must be implementable in hardware. A hardware implementation ensures fast computations, offloads the resources and ensures better security, as hardware is harder to attack than pure software. A simple XOR encryption[2] is light and easy to implemented in hardware. It is fast and messages will be encrypted in the same clock cycle [WES08]. Though, XOR offers not much protection, as it is vulnerable to known-plaintext attacks[3]. AES is a strong symmetric algorithm, provides a strong protection and can be implemented in hardware as well [HAHH06]. The downside of encryption, and

[2] XOR encrypt a binary plain-text by xor it with a repeating binary key, e.g. by XOR the key 1010 on the plaintext 1101 0011, we get the ciphertext 0111 1001.

[3] In known-plaintext attacks the intruder knows the ciphertext and some part of the plaintext. He can then reverse the XOR and get the key.

special strong encryption, is the increased computation time. By outsourcing the key creation to special hardware components with dedicated partitions and by ensure that a key will be available at the point in time a message has to be encrypted. In the software component, the message only has to be encrypted and sent (without concern of key generation or key management), and the increased computation time would be minimal. So even though an increased WCET must be taken into account, the limitation of a hard real-time behaviour will not be affected. Due to the strong protection of the AES algorithm and the minimal increasing in the WCET, as the keys are created and managed by hardware, I recommend AES for the encryption in the SC.

3.1.2.3 Crypto Component

If long lasting encryption is needed, a special *Crypto Component* can be implemented in the design as a supplement or addition to the SC. The Crypto Component will provide both the encryption and decryption and is designed as an ordinary component with TISS and host. The cryptographic services are carried out in the application computer. Where the SC only encrypt the message in the transport and decrypt the message at end-destination, a Crypto Component can apply cryptographic algorithms to information that is going to be stored in other tasks, i.e. a low CL task can obtain encrypted information without getting knowledge of its content.

To encrypt information, the Crypto Component receives a message from another task. It encrypts the message and forwards the encrypted message to a receiving task. The receiving task can be the same task that requested the encryption. The same procedure applies for decryption. After encryption the information are applied the lowest possible CL, regarding the original CL, and the IL is remain unchanged. It is important to note that the original CL and IL of the information must be restored after decryption. The CL at the receiving task must be higher or at the same level as the original CL of the information.

The Crypto Component can be designed similar to the SC with special hardware components to compute and manage the cryptographic keys. The encryption and decryption will take longer time than the SC, as the Crypto Component is a separate component and must have its own timeslot in the schedule.

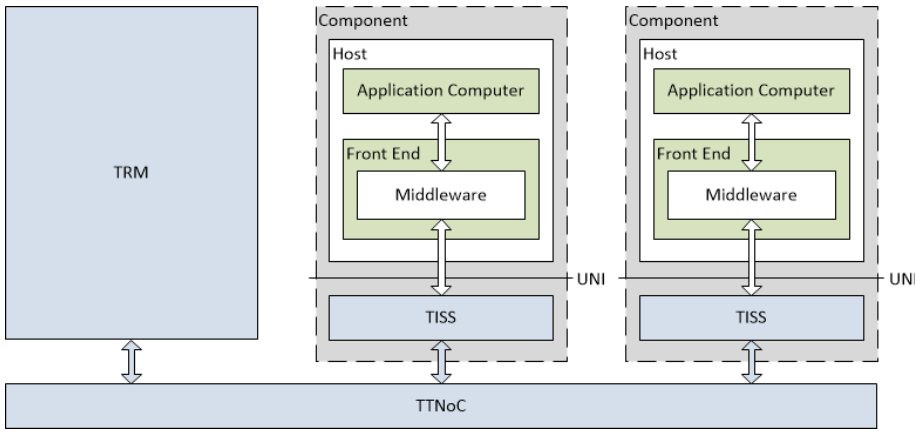

Figure 3.3: A simple version of the ACROSS MPSoC architecture. The two components is connected through the TTNoC network and the TRM are managing the communication routes. The TSS consisting of the TRM, the TTNoC and the TISS are coloured blue.

3.2 Architecture Examples

I will give tree simple examples of the architecture, the validation middleware (VaM) and the partitioning by a system of mixed criticality.

3.2.1 The ACROSS MPSoC Architecture

Figure 3.3 shows the ACROSS MPSoC architecture with two components. The figure is colour coded for easy recognition and the elements of the TSS are shown with blue colour. The components are of a gray colour and with a dashed frame. The hosts are in white and contains of AC and FE. Middleware are placed in both FEs. The task in the components communicates via the TTNoC by sending messages. The UNI will provide a transparent interface through the TSS. The component cannot change the time schedule in the TISS, but can suggest the TRM to reconfigure the time schedule and routes. If the change does not conflict with the safety and security restrictions, the TRM can change the configuration of the TISS. The middleware can be implemented in the FE and contain additional security functions for sending or receiving messages and function as an extension to the TISS. The FE also provides a *Port Memory* which houses the ports of the encapsulated communication channels [PPES09]. The port memory is not pictured.

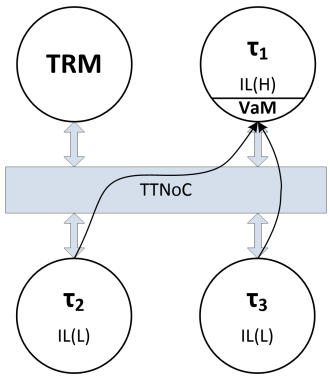

Figure 3.4: The VaM is placed together with a task τ_1 with a high IL. The VaM require input from redundant or diverse tasks, τ_2 and τ_3, with lower IL. A voting algorithm among the input in the VaM produces a high IL output to τ_1.

3.2.2 Simple VaM Example

Figure 3.4 illustrates a simplified version of the system containing two redundant task, τ_2 and τ_3, with a low integrity level, sending messages to the high level task τ_1. To allow an upward integrity flow a VaM is placed in τ_1 to collect the redundant low level input. The VaM runs a voting algorithm among the two input and produces a output with a high level value to τ_1.

3.2.3 The Partitioning

An example of partitioning of a system of three applications, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, with 13 tasks is illustrated in Figure 3.5. The tasks are mapped to on two PEs; N_1 and N_2 and are colour coded by their IL and CL configuration (see Table 3.1), e.g. τ_{13}, τ_{22} and τ_{34} have the same purple colour.

As discussed in Section 3.1, tasks of same configuration, i.e. same colour in this figure, can be placed into the same partition. This is done by τ_{13} and τ_{22}, and τ_{14} and τ_{23}. An exception is the two redundant tasks τ_{31} and τ_{32}, as they share an edge in the protection requirement graph, Π. The middleware is a part of the task and τ_{33} contains therefore both the application specific services and the VaM. Parts with gray shading are not occupied by a partition slice and remain unused. Tasks cannot start until they have received all its input information, e.g. τ_{16} need the information from τ_{14} and τ_{15} before it can start. A task output

Table 3.1: The colour coding of the tasks in Figure 3.5.

$IL(\tau_i)$	$CL(\tau_i)$	Colour
1	UC	Red
2	UC	Salmon
3	UC	Yellow
2	S	Purple
1	TS	Green
2	TS	Blue

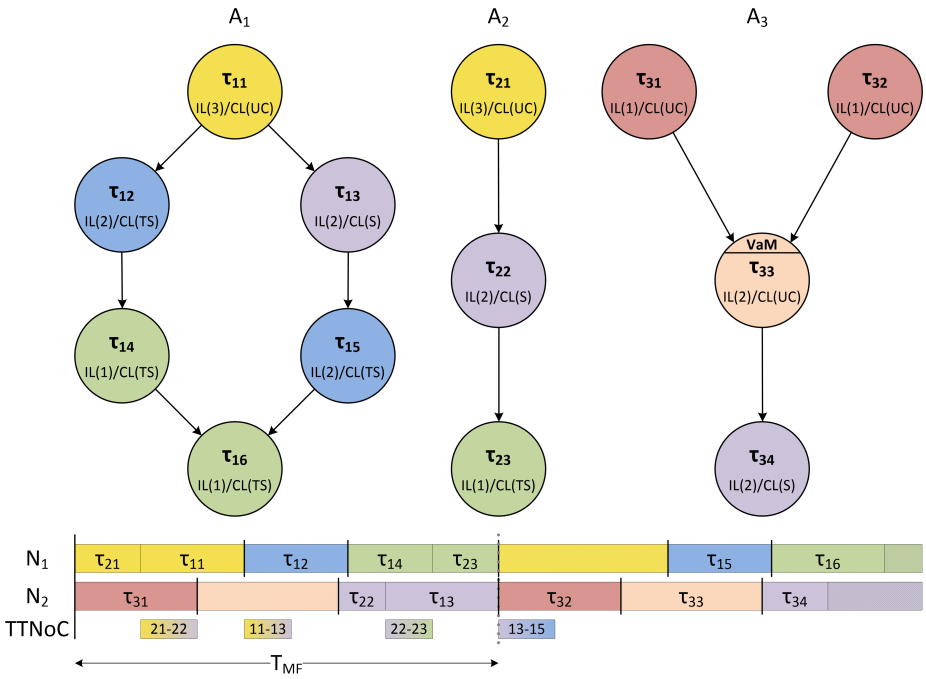

Figure 3.5: Partitioning of three subsystems; $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. The partitioning is not optimised, but reached the deadline before the period ends. The colouring indicated the same configurations of IL and CL and tasks of same configuration can share a partition. An exception is τ_{31} and τ_{31}, which has the same configuration, but are redundant tasks and prohibited to spare partitions. τ_{31} and τ_{31} share an edge in Π. The messages is colour coded by the configuration of the sending and receiving task for easy visual identifying.

information after it is terminated.

Both the downward integrity flow and the upright confidentiality flow are met. In $\mathcal{A}_2$ the information flow is clear and information flows from τ_{21} with a high IL and a low CL to τ_{23} with a low IL and high CL. In $\mathcal{A}_3$, the two redundant tasks τ_{31} and τ_{32} have a upward integrity flow to τ_{33}. A VaM is implemented in τ_{33} and runs a voting algorithm among the two input and produce an output of a higher IL.

3.3 Safety Mechanism

The ACROSS MPSoC architecture is developed to support integrity and to be easily certified. A lot of mechanisms are build into the system to ensure integrity. I have made no extension to the model to increase safety.

3.3.1 Separation and Partitioning

A safety system must guarantee the safety ability through validation. Without certification the system cannot claim to be safe. In complex system the certification can be a hassle. The separation of subsystems [Rus81] makes the certifications easier, as the subsystems can be certified separate and not as one. The separation also enforces the safety in the partition layer. The SK separates the partitions both in the spatial and temporal domain. This prevent information in one partition from flowing unintended to another partition and thereby enforcing the information flow, as discussed later on in Section 3.3.4.

The partitions are sanitised by the SK. This means that no old information would be left in the partition to be revealed from one task to another, i.e. information cannot flow from one task to another, just because they are using the same partition at different time.

Partitioning also provides damage limitation, so fault in one partition will not affect other partitions. The safety aspect of this is immediate, as a fault in e.g. a sensor would not affect the safety of the system other than the missing or faulty output from the sensor. As it is possible for tasks of same configuration to share a partition and thereby be affected of the same damage in a partition, it is important for the designer to consider if some tasks should be prohibited to share a partition. In Section 3.1 there is a discussion about the requirements for separation.

3.3.2 Redundancy / Diversity

Allowing a flow from low integrity tasks to high integrity task, is the premise for including redundant tasks. Redundancy enforces safety in two ways: (1) it makes the system fault-tolerant, as if one redundant device fails, the other device(s) would probably not fail of the same reason. For that reason it is important redundant tasks does not share a partition. (2) the redundancy relaxes the rigid information flow, as redundant tasks can be validated to a higher level and redundancy makes the system easier and cheaper to certify, because redundant tasks have a lower level than one single task. Redundancy is often carried out by hardware components. Diversity, where the same functionality is computed using different algorithms and likely by different development teams, is often preferred in software. This minimises the risk of a software bug in one task to occur in another diverse task.

3.3.3 Time-Triggered Architecture

The ACROSS MPSoC is a *Time-Triggered Architecture* (TTA) and ensures reliable and trustworthy hard real-time communication. It guaranties a sending slot to all tasks in a cyclic period and at an a priori known time. Only in that timeslot a given task can send its message. At the same time, the receiving task knows it has to receive a message. This prevent a flood of information in the system that could potentiality make the system to malfunction in a non-safety manner. The TTA enforces the properties of the SK and can be considered as a realisation of the SK [WESK10].

3.3.4 Trusted Subsystem

The TSS ensures safety by transparently manage the communication between the tasks. A task cannot change the TSS but has to suggest a reconfiguration. Only if the safety is still guaranteed, the change will apply. This makes the TSS quite robust for unauthorized changes that could lower the safety of the system.

The encapsulated communication channels are unidirectional channels with a single sender and one or more receivers. The channels with its endpoints and it temporal presence, are known a priori and constantly checked by the TRM [WESK10]. The encapsulated communication channels and TRM are both part of the TSS. The encapsulation guaranties the information flow, i.e. there can be no flow from a low IL task to a high IL task (unless it runs through a validation

middleware). This prevents unsafe elements to be unintended promoted.

3.4 Security Mechanism

Security covers both integrity and confidentiality. The security integrity is quite similar to the safety integrity, but focus on protecting the system from malicious attacks, i.e. protecting the system from the environment. The same mechanisms that enforce the safety integrity also enforce the security integrity and I thereby adopt the integrity approaches from the safety architecture. For confidentiality, the essential is that no information is revealed. Even though the ACROSS MPSoC architecture is focussing on the integrity, lots of the properties provided also enforce the confidentiality.

The security mechanisms provide protection as a Lattice-based access control with one label [San93], i.e. if you have the right security level according to the IL and CL, you will be allowed to read or write information. In contrast Role-based access control [SCFY96] could provide a more complex security, as access control are assigned to different roles instead of just a security level controlled by the system. Role-based access control will not be taken into account or covered by this paper. That means that we treat information as it has no special owners or readers, but just a specific level of security.

3.4.1 Individual Integrity and Confidentiality Level

To ensure the confidentiality we have added a confidentiality level to the original design. The CL prevents the information from flowing from a high classified component to a lower classified component [WESK10]. The confidentiality level is not merged into the integrity level to obtain one single combined security level, but is an independent security level. This ensures the information flow in a secure and orderly manner.

I have earlier in Section 2.3 and 3.1.2 discussed the relation between the integrity level and the confidentiality level and illustrated it in Figure 2.2. A more complex figure with four levels of integrity, where IL 1 is lowest and IL 4 is the highest, and three levels of confidentiality (UC<S<TS) are shown in Figure 3.6. Component τ_1 has a high integrity level and a low confidentiality level and can therefore send messages to all the other components. In the other end of the scale is τ_4 with a low integrity level and a high confidentiality level. τ_4 can only send messages to other components with the same level configuration, but as no

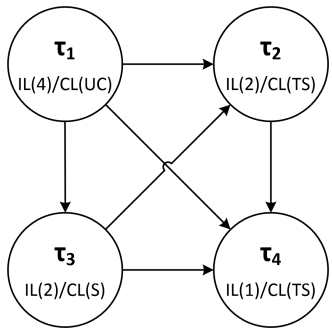

Figure 3.6: This is a more complex configuration than Figure 2.2. In this example the tasks can have an IL from 1 to 4 and a CL of either UC, S or TS. It complicates the information flow, but still the combination of the highest IL and lowest CL in τ_1 has a free flow to the other tasks.

such exists, it cannot send messages to the other components. τ_2 has a medium integrity level but a high confidentiality level, can therefore only send messages to τ_4; and τ_3 has a medium level in both integrity and confidentiality and can send messages to τ_2 and τ_4.

3.4.2 Separation and Partitioning

The separation is an architectural foundation in the ACROSS MPSoC architecture. By isolating the tasks of different criticality, the integrity as well as the confidentiality are enforced, since information cannot be leak or damage other partitions. Several tasks of same security configuration can share a partition and the information inside a partition can potential flow from one tasks to another. This will not be a problem, as information is not bound to an owner but to a mutual security level and the tasks sharing the partition are at the same level. If needed it is possible for the designer to prohibit tasks to share a partition.

The separation does not enforce the confidentiality if an intruder gets access to the information. But is prevent a covert channel, where information are unauthorised leaked from one partition to another, to be formed. Neither can old information, placed in a partition by a high CL task, be read at a later point in time by a low CL task, due to the sanitation of shared partitions.

3.4.3 The Trusted Subsystem

The mechanisms that provide safety in the TSS also provide security, as the encapsulated communication channels improve both the integrity and confidentiality. It controls the information flow and guaranties the identity of the sending and receiving component through the TRM. This guaranty of the identity, are a security property. It ensures that a malicious component cannot take the place of another component and thereby send malicious information or receive sensitive information. It also enforces the safety, as only authorised components can send messages and no malicious information can endanger the safety.

3.4.4 Secure Channel

Confidentiality is ensured by the flow control as long as an intruder does not get access to the flow by e.g. eavesdropped on a communication channel, i.e. the information are only secure as long nobody gets access to it. To protect the information from unauthorised access we can encrypt the information and make it gibberish for the intruder. A SC adds cryptographic algorithms to the information from end-to-end sent over the TTNoC [IW13]. The AES algorithm is recommended. The Secure Kernel of the SC can be implemented in hardware and manage the shared keys and key generation on a special hardware component. As it is the key generation that takes the most computational power, and applying the key to the information only increase the computational time minimal, the SC ensures a strong and fast en-/decryption to the system. This enforces the secrecy and confidentiality. The SC does not affect the normal confidentiality flow.

3.4.5 Crypto Component

For a longer lasting encryption, a Crypto Component can be used. The Crypto Component is a special component that adds cryptographic algorithms to information, but is not decrypted at the receiving task, as is the case with the SC. This ensures that the task cannot understand the information and thereby reveal the secret information.

The Crypto Component ensures that output will have the same IL as the input and only task with the same (or higher) CL can have the information decrypted again, i.e. encrypted information will have the lowest CL, but return to its original (or higher) CL when decrypted. This re-establishment of the CL is

important to ensure the confidentiality flow.

3.5 Behaviour

Without the extensions to the communication services provided by middleware, the information flow would be quite rigid. The VaM relaxes the integrity flow by allowing redundant low IL tasks to send messages to a high IL task with a VaM implemented.

Communication between two SoC uses a external network. Information on that network is target for eavesdropping and needs to be encrypted in order to ensure the confidentiality. A Secure Channel is implemented in the middleware and handles the encryption in a fast and efficient manner.

To understand those two mechanisms I give two behavioural examples and follow a message from the sending task to the receiving task.

3.5.1 VaM

Figure 3.7 describes the message path from task τ_1 in component C_1 to task τ_4 in component C_4. τ_1 have a lower IL than τ_4, so a VaM are implemented in C_4. τ_1 writes a message m_1 (1) into the port memory (2) in the FE. The schedule S_S in the TISS (3), control when the message from the port memory (4), can be sent out on the TTNoC (5). The mechanisms and routing in the TTNoC will not be discussed here and can be interpreted as a black box managed by the TRM. When the message arrives at the receiving component C_4 (6), the schedule S_R (7) in the TISS will check that the message m_1 arrive at the correct time from the correct sender. As the sending task τ_1 has a lower IL than the receiving τ_4, the message goes through the port memory (8) to the VaM (9). In the VaM the messages have to wait until redundant/diverse messages arrives from the redundant/diverse tasks τ_2 and τ_3 in components C_2 and C_3. The VaM runs a validation algorithm when all messages have arrived, to produce a higher IL output. The new message m_2 (10) is received by the task τ_2, which now can start to provide its application service.

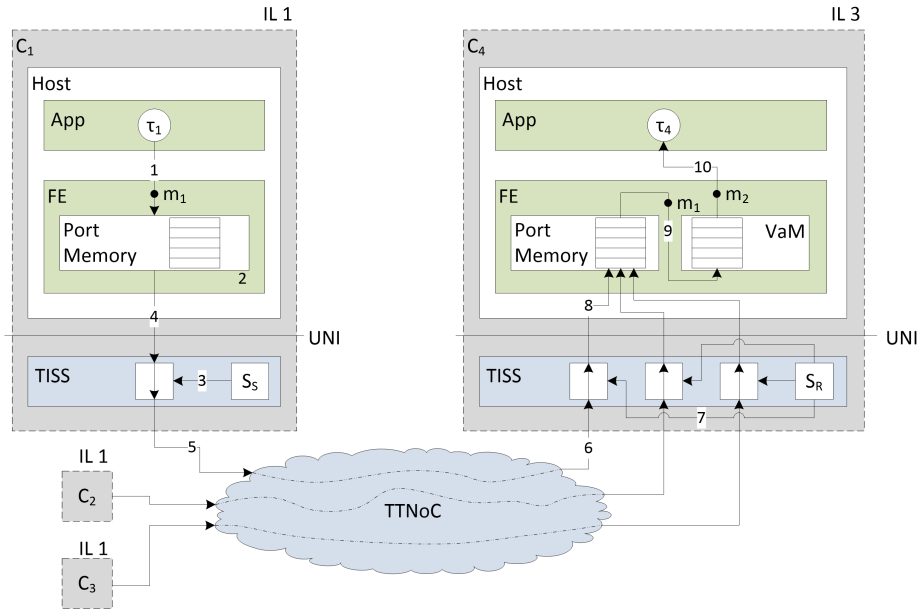

Figure 3.7: Illustrates the behaviour of the VaM. The message path for m_1 with IL 1 is followed through the system. The VaM takes three redundant messages, arriving at different timeslots, to produce a new message m_2 with IL 3.

3.5.2 Secure Channel

Figure 3.8 describe the usage of the Secure Channel (SC) and the gateways in IO-components. I will not take IL or CL into account as it brings nothing to the understanding of the behaviour of the system when using a SC. The description is of the system and not of the SC itself nor is the behaviour inside the SC described.

A task τ_{11} placed in C_{11} want to send a message m_2 to τ_{12} in C_{12} and a message m_1 to τ_{22} in C_{22}. τ_{11} and τ_{12} is placed on the same SoC and uses therefore the standard communication channel and is comparable to the behaviour described in VaM. The standard communication channel offers no encryption and will not increase the WCET for the tasks. It is used in on-chip communication, as it is assumed eavesdropping on the TTNoC is not feasible.

I will in the following simplify the description of the behaviour in the FE and refrain from describing the standard communication channel, as it has already

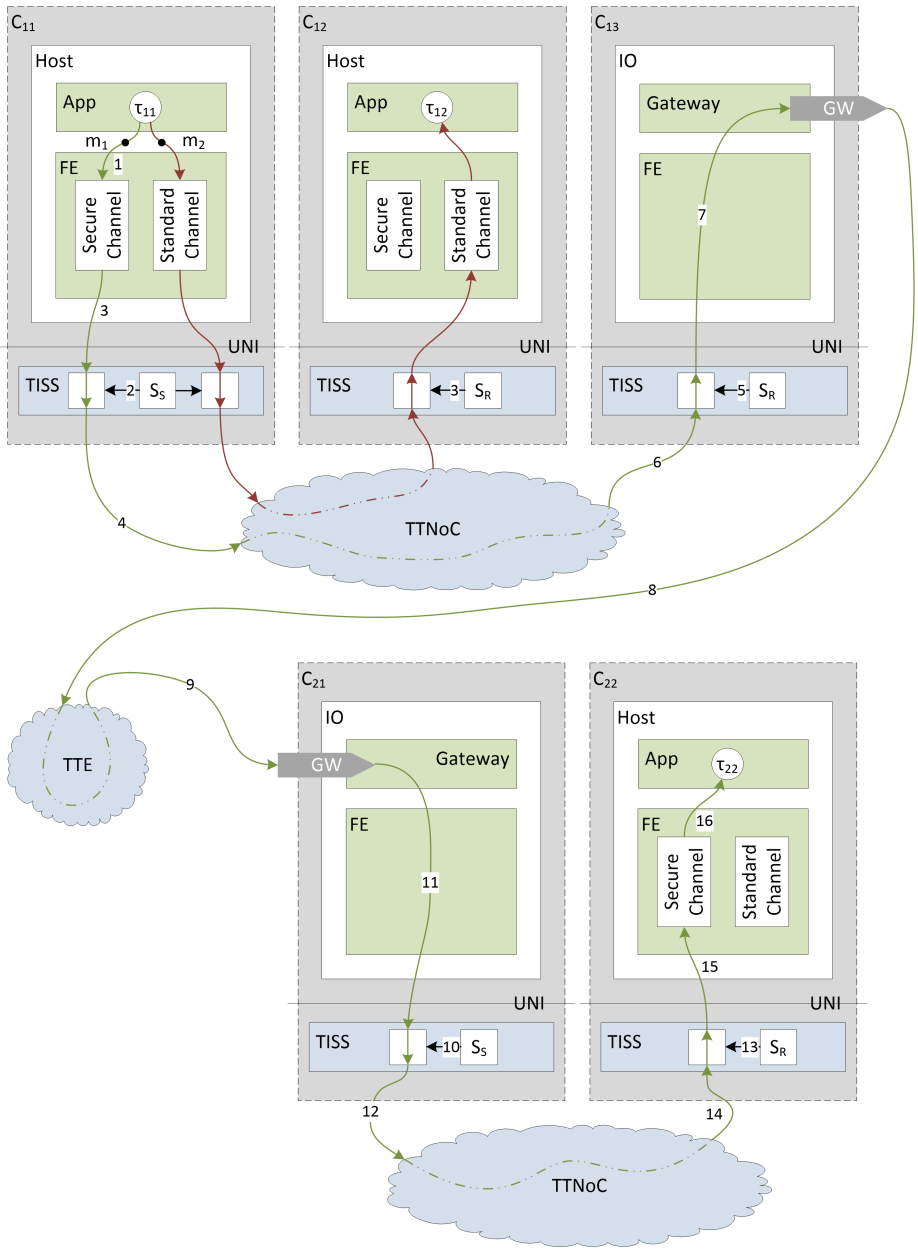

Figure 3.8: Illustrated the behaviour of the system with a SC implemented. The red message path shows the message m_2 using a standard communication channel in an on-chip communication between τ_{11} and τ_{12}. The green message part for m_1 is using the SC to establish off-chip communication between τ_{11} and τ_{22}. A TTEthernet are used to combine the two SoC.

been done for the VaM in Section 3.5.1. For the same reason will I not describe the message path for m_2 (the red path) and only describe the path for m_1 (the green path), as τ_{22} are placed on another SoC and m_1 thereby goes through the Secure Channel. The two SoCs are linked together with an external network, in this case a *TTEthernet*.

The message flow (1) from τ_{11} to the SC where it is encrypted with a symmetric key. The key management and generator are outsourced to a dedicated hardware component and are transparent to the tasks. The a priori known temporal schedule S_S (2) in TISS allows message m_1 (3) to flow to the internal TTNoC (4). An equivalent receiving scheme S_R (5) allows message m_1 to arrive (6) at the specialised IO-component. A gateway is placed in the IO-component, providing an interface to the external network. The encrypted message (7) flows through an output gateway and out to the external network (8). An IO-component in the receiving SoC (9), receives the message through an input gateway. The message flows through the new SoC (10-13) in the same way described in (2, 4-6) and is received by C_{22} (14). The message flows through the SC (15) and finally (16) ends it flow at τ_{22}.

3.6 Assumptions

The paper is purely theoretical and some assumptions are important for the design of the system. The first three assumptions are crucial for the safety and security of the system, while the last two have more character of relaxing the description.

3.6.1 TSS Is Free For Design Faults

Even though it is common knowledge that no software is free for bugs, it is assumed that the *separation kernel* (SK) and TSS are free for bugs and design faults. To realize this assumption, the trusted parts (the SK, TISS, TRM and TTNoC) are designed as simple and with as small piece of code as possible. This makes them easy to analyse and validate. Furthermore the TSS is transparent to the architects of tasks, who programs up to an interface with no possibility to change the TISS. If SK or TSS is compromised, the foundation of the safety and security of the architecture collapses.

3.6.2 No Malicious Attack Before Runtime

As an addition to the previous assumption, it is assumed that a malicious attack has not taken place in the design phase, i.e. the TSS is free from design faults caused by both the designer and a malicious attacker.

Possible attack in the design phase could be to add a backdoor to get access to the system at runtime or implement a covert channel to obtain information of the system. Virus could also be implemented by malicious persons, causing the system to misbehave or leaking information. Even though the threat is real (a virus on the computer the system are developed on, could implement itself to the TSS), these attacks in the design phase are assumed not to happen and therefore it will not be covered.

3.6.3 No Malicious Attack on TTNoC

It is assumed that it is not feasible to attack the TTNoC or reading the memory of the components [IW13]. However is it possible to manipulate or eavesdrop upon the physical connections and the pins on the physical device. Combined with assumption of no attacks before runtime, the SoC cannot be tampered if the SK and TSS works as intended.

3.6.4 Not Looking Into the TTNoC

I regard the TTNoC as a closed and trusted system, i.e. a black box. I have looked at the time-triggered properties but not how it works in details or how the messages are routed inside the TTNoC. The reader is redirected to [SEH+12] where the TTNoC are described in some details or to [Sch07] for deeper details.

3.6.5 One Combined Safety-Security-Level

Safety integrity and security integrity are both interpret as one combined safety-security-level. By that the integrity level (IL) covers both safety and security in one label. It is allowed as both safety and security wants to protect alternation and a violation against one kind of integrity also violates the other kind of integrity. Furthermore the decomposition of safety integrity would not affect the security integrity.

Chapter 4

Design Tasks

In this chapter I will try to elucidate the consequences a design decision will have on the system.

4.1 How design decisions influence the system

For every action there will be a reaction, as Newton's third law says. This applies at some point to the architecture of the system. We must make some considerations and decisions to improve the system, but those decisions may cause some restrictions. By improving the safety or security of the system, the cost in development time and money will increase. And a use of strong security on a real-time safety system, the computation of messages may be slowed so much down, that it may affect the real-time and thereby the safety.

4.1.1 Safety

The problem the ACROSS MPSoC had to solve, was the difficult certification of large and complex safety systems. Earlier integrated safety systems, as Totel's

integrity model [TBDP98], were certified as a whole at the highest integrity level of the system. The ACROSS MPSoC solve this problem by the concept of separation of subsystems [WESK10], so that the subsystems could be certified separately. As certification means time and money, the cost of a ACROSS MPSoC system was cheaper than earlier systems. Even though, raising the safety of a system, by increasing the integrity level of a subsystem, still means a system is certified at a higher level and thereby at a higher cost.

A task can be decomposed to two redundant tasks with lower integrity levels according to Table 2.3. This means the cost to certify the system will decrease, but also means there are more tasks to be placed on the partition slices and the schedule table. A Tabu Search-based approach is proposed in [TSP13] for optimisation and calculates the optimal configuration to lower the development cost. Without the optimisation the task would be hard to schedule. To schedule tasks without optimisation the period could be extended, but that is not a solution and could compromise the hard real-time properties and thereby the safety. Due to the longer $\mathcal{T}_{cycle}$ a single task would be assigned access to the communication network less often and would not meet its deadline.

Even though a lower price in the development is desirable, it come with a cost in the unit price, as a lower cost in the design phase caused by decomposition of tasks, will result in more units. A higher unit cost is probably more affordable and preferable than a high development cost.

The safety of the system can also be increased by using more redundancy on the SoC, but it would increase the number of tasks and units as discussed above. Redundancy of the SoC, with off-chip communication is alto a possibility, but is out of the scope of this thesis. The reader is redirected to [OKS08].

4.1.2 Security

Security can be crucial to ensure safety. Adding strong security to safety systems may compromise the safety properties, e.g. longer WCET caused by heavy security computations. It is important to remember that the system fundamentally is a safety system with a layer of security and not the other way around.

4.1.2.1 Encryption

Encryption will add secrecy and thereby some confidentiality to the safety system. It is easy to apply, as it does not interfere with the integrity flow. But

encryption is not just encryption. The level of security can vary from light encryption, easy to compute but also easy to break, to strong encryption which demand heavy computations but is hard (if not impossible of today's standard) to break. Embedded systems have limited computational resources and strong encryptions could start a race for resources, so encryption is a payoff between powerful secrecy and fast computation.

Light encryption, e.g. XOR, is less invasive than strong encryption and need less computational power. With strong encryption, e.g. AES, follows heavy computations, which costs a lot of resources and is time demanding. The simulation in [WES08] shows us that AES is harder to compute than XOR. Most of the hard computations is creating and managing the keys. The secure channels proposed by [IW13] offloads the key generation and key management out of the SoC to special system components. By doing so and by ensuring that a key will always be ready to use when needed, the temporal extension will be minimal, as the key only has to be applied to the message. This make the encryption less invasive, but a little increment in the WCET must be taken into account and the schedulability will decrease a bit.

4.1.2.2 Flow and Confidentiality Levels

Encryption are only one part of security confidentiality, the other part are the confidentiality flow. The upward confidentiality flow clashes with the downward integrity flow. For that reason every task has two independent security levels, the IL and CL. A confidentiality flow interfere therefore not with the integrity flow. However a high CL will interfere with the overall information flow, as a low CL flows freely to other CL, while high CL is restricted to flow to other CL at the same level. Adding a unnecessary high CL to a task could cause problems in the information flow and the designer must carefully decide the CL for each task. E.g. if redundant sensors are assigned a unnecessary high CL, the receiving task must be at a equally or higher CL.

A system with lots of high CL tasks might not necessary be a problem to the schedulability due to restrictions of the information flow. If all tasks are at the same level, the information can flows freely if integrity is not taken into account. But a security system with only a single level of security would not need CL at all and could just adding encryption. A widely variation of the CL (and IL) can on the other hand decrease the schedulability, as the dependencies became more complex. The information between the tasks is given before the scheduling, but it is important that task placed in the same partition is at the same CL. Elevation of a task to a higher CL is not possible as it is with IL.

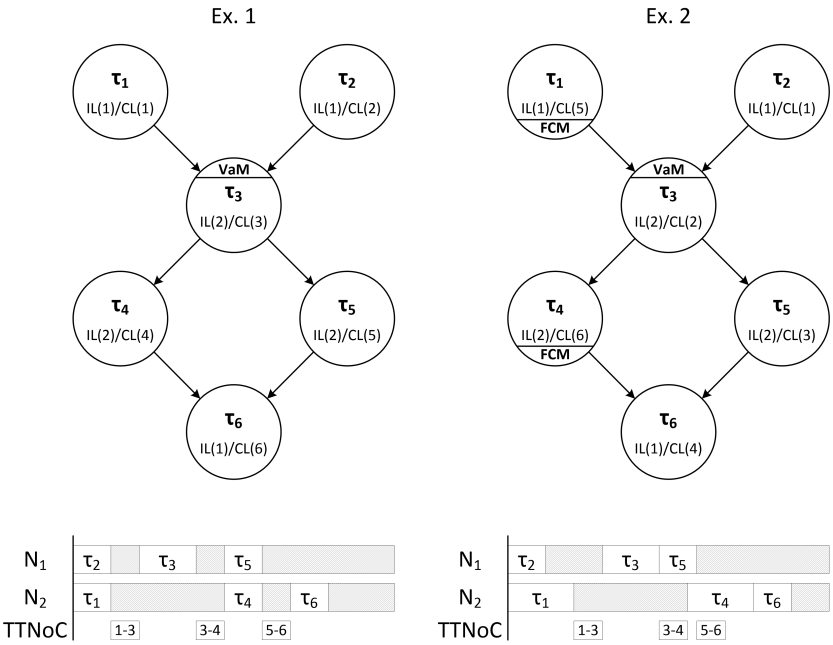

Figure 4.1: A toy example of the increasing WCET by using middleware, caused by a poor decision in the assignment of CLs. In Ex. 1 the information have a clean upward confidentiality flow, while in Ex. 2 the τ_1 and τ_4 need the implementation of a FCM in order to circumvent a downward confidentiality flow. The use of middleware increases the WCET causing the deadline to be delayed.

The levels in CL are not limited to a specific number and so far in this paper I have operated with three levels; UC, S and TS. It is possible to apply each task with its own unique CL. This will improve the security as we will have a strict unidirectional confidentiality flow through the system. With unique CL tasks cannot share partitions, witch also improve the security, as information cannot flow between two tasks in separate partitions, unless they communicate through the conventional communication routes.

With unique CL for each task, the designer need to have a complete overview of all the tasks. In some cases encryption or a flow control middleware (FCM) must be applied in order to circumvent the upward information flow, but both solutions will extend the WCET of the tasks, as a task covers bothe the application specific services and the middleware. A toy example to illustrate the extended WCET is given in Figure 4.1. Here *Ex. 1* illustrates a system with clean upward confidentiality flow and *Ex. 2* illustrates an identical system, but

with a poor decision of assigning CLs, causing a downward confidentiality flow from τ_1 to τ_3 and from τ_4 to τ_6. To circumvent the flow a FCM are added to τ_1 and τ_4, which resulting in increased WCET for the two tasks.

4.1.2.3 Optimisation

The Tabu Search-based optimisation algorithm proposed by [TSP13] is not designed for confidentiality. The addition of the confidentiality to the system has a direct impact on how to schedule the tasks and messages. An additional computation time for middleware and encryption increases the WCET, the upward information flow must be taken into account and the prohibition of two tasks with different CL sharing a partition; must all be added to the algorithm.

4.1.2.4 Security or Safety

Our system offers security decisions made in the design phase of the systems. When the architecture is in use, it is not feasible to change or update the security. At update, the system must be taken offline. I have assumed that the trusted part of the system is free of design faults and no malicious attack has taken place before runtime. In reality such assumptions might not hold, as malicious code can be smuggled into the system at the design phase or via an update of the system. A virus or a Trojan horse can be added and endanger the safety and security of the system. To react upon such malicious code, we need some updated antivirus. But it is a endless race with the malicious code on one side and the antivirus on the other, as threats are freshly discovered all the time and the scanner must be updated to recognise them.

This brings another problem. We can guarantee the safety *or* the security, but not both. Safety requires slow updates to validate the system and ensure it is still safe. Security needs quick updates to ensure the antivirus to recognise the threats. The safety validation could easily take weeks or months, and in that time the system is exposed for the known threads. Even new threats may be discovered during the validation and the system needs to be updated and validate once again. In other words, if the system is validated caused by a security update, it is not completely secure during the time the validation takes *or* if the system is updated without a validation, it might not be safe. This leaves us with the question: "Do we want our system to be completely safe or completely secure?", because we cannot have both. Due to the assumptions of no such attack, this is out of scope of this thesis, but could be a future research project. The system I propose is a safety system, with some security properties

in form of secret communication and unidirectional information flow between tasks in relation to their safety and security configuration.

4.1.3 Schedulability

The ability to schedule the tasks is important for the system. Decisions made for safety or cost can increase the number of total tasks and that will affect the schedulability. It could happen by introduce many redundant tasks for safety reasons or by deduction to reduce the cost.

Tasks can share partitions if the configurations of the IL and the CL are the same and the tasks not sharing edges in Π, as described in Section 2.1. The more the tasks can share a partition, the easier it is to schedule. To increase the schedulability, a task can be elevated to a higher IL. By elevating the task it can be placed in the same partition as other tasks with the same IL and are thereby easier to schedule. The cost will increase with elevation.

Figure 4.2 gives a example of five tasks and the sharing of partitions. The tasks IL and WCET are shown in Table 4.1 and have no input nor output. The processing element N_1 has two partitions in a MF; one of IL 1 and one of IL 2. In a) τ_5 is not mapped to the N_1, as τ_5 is of IL 1 and the free space in the partition on N_1 is of IL 2. In b) τ_5 is elevating and it can share partition with τ_4. A toy example are illustrated in c) where all tasks are of IL 2. The tasks can now share all the partitions, and make the scheduling easier for the designer.

Decisions made for security may affect the schedulability as well. The requirement of the same CL among tasks sharing the same partition decreases the schedulability. The restricted information flow makes the system more complex with fewer possible communication paths, but those paths are designed before the scheduling and will not affect the schedulability.

To increase the schedulability, it is desirable to have as many equal tasks with the same safety and security configurations as possible. But it would in most cases increase the cost and would not be desirable for the safety or security requirements.

Table 4.1: The configuration of the tasks used by Figure 4.2.

τ_i	$IL(\tau_i)$	$WCET(\tau_i)$
τ_1	1	2
τ_2	2	2
τ_3	1	2
τ_4	2	1
τ_5	1	1

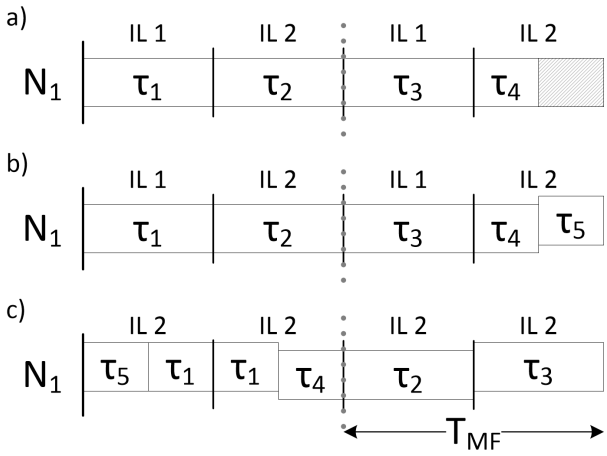

Figure 4.2: Illustrates the the scheduling of the tasks from Table 4.1. In a) τ_5 cannot be mapped as there is no free partition for IL 1. In b) τ_5 is elevated and can share partition with τ_4. c) is a toy example where all tasks are elevated to IL 2 and can share partitions with each other and makes it easier to schedule.

Chapter 5

Conclusion

I have in this thesis discussed some methods to add security to a safety system. The security layer is applied the ACROSS MPSoC architecture, which is based on integrity and therefore already have security integrity applied.

Adding confidentiality to an integrity system, need some consideration and design decisions. To overcome the challenge of two opposing flow, I have present a design that allows both an integrity flow and a confidentiality flow. A task has already been applied an integrity level in the ACROSS MPSoC architecture, which determine the downward flow in the integrity domain. By adding a supplementary confidentiality level determining the upward flow in the confidentiality domain, a task can be configuration with restrictions both the integrity and confidentiality flow. The two levels are not affecting each other.

Confidentiality covers the secrecy as well, for what I have suggested the use of a secure channel. By applying end-to-end cryptographic algorithms to information going outside the SoC, an eavesdropper would not be able to comprehend the leaked information on an external network. I suggest the usage of AES cryptographic algorithm, as it can be implemented in hardware and ensure a minimal increase in computation and WCET.

The usage of a special dedicated component for cryptographic, has been discussed, as a supplement or replacement for the secure channel. It has the ability

to provide long lasting encryption compared to the end-to-end approach of the secure channel, but also creates more messages and tasks to the system. The importance of recreate the security level configuration of an encrypted message, is also discussed in this paper.

I have made some pointers to consider for the partitioning, the schedulabitity and for adjusting the Tabu Search-based optimisation algorithms, in order to take the confidentiality into account. The confidentiality level of a task has a direct impact on how to share partitions. The extra time in the WCET caused by encryption, must also be taken into account.

For future work it could be interesting to consider attack before runtime and analyse the issue of a security system with a short update interval, combined with the slow certified safety system. Another topic for future work could be to investigate how Role-based access control, in contrast to the Lattice-based access control used in this thesis, would affect an embedded safety system.

Appendix A

Abbreviations

Application Computer	*AC*
ARTEMIS CROSS-Domain Architecture	*ACROSS*
Component	*C*
Confidentiality Level	*CL*
Evaluation Assurance Level	*EAL*
Flow Control Middleware	*FCM*
Front End	*FE*
Integrity Level	*IL*
Multiple Independent Levels of Security (and Safety)	*MILS*
Multi-Level Object	*MLO*
Multi-Processor System-on-a-Chip	*MPSoC*
Processing Element	*PE*
Resource Management Authority	*RMA*
Secret (level in CL)	*S*
Secure Channel	*SC*
Static-Cyclic Scheduling	*SCS*
Safety Integrity Level	*SIL*
Separation Kerne	*SK*
Single-Level Object	*SLO*

continues...

System-on-a-Chip	*SoC*
Trusted Computing Base	*TCB*
Time-Division Multiple Access	*TDMA*
Trusted Interface Subsystem	*TISS*
Trusted Network Authority	*TNA*
Trusted Resource Manager	*TRM*
Top Secret (level in CL)	*TS*
Trusted Subsystem	*TSS*
Time-Triggered Architecture	*TTA*
Time-Triggered Network-on-a-Chip	*TTNoC*
Unclassified (level in CL)	*UC*
Uniform Network Interface	*UNI*
Validation Middleware	*VaM*
Validation Object	*VO*
Worst Case Execution Time	*WCET*

Appendix B

Notations

A set of all the applications	Γ
An application subsystem in Γ	$\mathcal{A}_i$
The application graph	$\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)$
A set of all tasks	$\mathcal{V}$
A set of all nodes	$\mathcal{V}_i$
A set of all edges in $\mathcal{G}_i$	$\mathcal{E}_i$
A task in $\mathcal{G}_i$	$\tau_i \in \mathcal{V}_i$
The mapping of tasks to partition elements	$M : \mathcal{V}_i \rightarrow \mathcal{N}$
Assignment of task to partitions	$\phi : \mathcal{V} \rightarrow \mathcal{P}$
Set of partitions	$\mathcal{P}$
Partition	P_j
Set of partition slices of P_j on N_i	$\mathcal{P}_{ij}$
The k^{th} partition slice of P_j on N_i	p_{ij}^k
A set of the processing elements	$\mathcal{N}$
An edge in $\mathcal{G}_i$	e_{jk}
A protection requirement graph	$\Pi(\mathcal{V}, \mathcal{E})$
τ_i and τ_j are not allowed in same partition	$sr_{ij} \in \mathcal{E}$
An message in $\mathcal{G}_i$	m_i
Size of m_i	s_{m_i}

continues...

WCET for task τ_i	C_i
Deadline for $\mathcal{G}_i$	$D_{\mathcal{G}_i}$
Period for $\mathcal{G}_i$	$T_{\mathcal{G}_i}$
Integrity level of τ_i	$il(\tau_i)$
Confidentiality level of τ_i	$cl(\tau_i)$

Bibliography

[AFHOT06] Jim Alves-Foss, W Scott Harrison, Paul W Oman, and Carol Taylor. The mils architecture for high-assurance embedded systems. *International journal of embedded systems*, 2(3):239–247, 2006.

[Alb04] Amos Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. page 235–252, 2004.

[BDRS08] C. Boettcher, R. DeLong, J. Rushby, and W. Sifre. The mils component integration approach to secure information sharing. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1.C.2–1–1.C.2–14, Oct 2008.

[Bib77] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, DTIC Document, 1977.

[BL73] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations, November 1973.

[CC12] Common criteria for information technology security evaluation. `http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R4.pdf`, September 2012. Part 3: Security assurance components.

[ESOHK08] C. El-Salloum, R. Obermaisser, B. Huber, and Hermann Kopetz. A novel naming scheme for system-on-a-chips supporting dynamic resource management. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 135–144, May 2008.

[HAHH06] P. Hämäläinen, T. Alho, M. Hannikainen, and T.D. Hämäläinen. Design and implementation of low-area and low-power aes encryption hardware core. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 577–583, 2006.

[HH09] Denis Hatebur and Maritta Heisel. A foundation for requirements analysis of dependable software. In *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security*, SAFECOMP '09, pages 311–325, Berlin, Heidelberg, 2009. Springer-Verlag.

[IW13] H. Isakovic and A. Wasicek. Secure channels in an integrated mpsoc architecture. In *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, pages 4488–4493, Nov 2013.

[KF09] Ronald L. Krutz and Alexander J. Fry. Mastering the certified secure software lifecycle professional. In *The CSSLP Prep Guide*. John Wiley and Sons, 2009.

[LABW92] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, November 1992.

[LRNT06] M. B. Line, L. Rostad, O. Nordland, and I. A. Tondel. Safety vs. security? (psam-0148). *PROBABILISTIC SAFETY ASSESSMENT AND MANAGEMENT*, page 151, 2006.

[OH11] R. Obermaisser and O. Hoftberger. Fault containment in a reconfigurable multi-processor system-on-a-chip. In *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, pages 1561–1568, June 2011.

[OKS08] R. Obermaisser, H. Kraut, and C. Salloum. A transient-resilient system-on-a-chip architecture with support for on-chip and off-chip tmr. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 123–134, May 2008.

[PPES09] H. Paulitsch, C. Paukovits, and C. El Salloum. Fault isolation with intermediate checks of end-to-end checksums in the time-triggered system-on-chip architecture. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 90–99, July 2009.

[Rus81] J. M. Rushby. Design and verification of secure systems. *SIGOPS Oper. Syst. Rev.*, 15(5):12–21, dec 1981.

[San93] R.S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, Nov 1993.

[SCFY96] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[Sch07] M. Schoeberl. A time-triggered network-on-chip. pages 377–382, 2007.

[SEH+12] C.E. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek. The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 105–113, Sept 2012.

[TBDP98] E. Totel, J.-P. Blanquart, Y. Deswarte, and D. Powell. Supporting multiple levels of criticality. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 70–79, June 1998.

[TSP13] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time systems. DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark, 2013. Technical University of Denmark.

[WES08] Armin Wasicek and Christian El Salloum. End-to-end encryption in the ttsoc architecture. In *Ebedded Systems Week*. ACM, 2008. talk: Embedded Systems Week 2008 (ESWEEK08), Atlanta, Georgia, USA; 2008-10-19 – 2008-10-24.

[WESK10] A. Wasicek, C. El-Salloum, and H. Kopetz. A system-on-a-chip platform for mixed-criticality applications. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*, pages 210–216, May 2010.

[WM12] Armin Wasicek and Thomas Mair. Secure information sharing in mixed-criticality systems. In *Proceedings of the World Congress on Engineering and Computer Science (WCECS 2012)*, volume 1. IAENG, October 2012.