ISEL
ADEETC
Área Departamental de Engenharia Electrónica e de Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Relógio de Ponto
(*Working Time Recorder*)

Projeto
de
Laboratório de Informática e Computadores
2020 / 2021 verão

publicado: 15 de março de 2021
atualizado: 12 de abril de 2021

1 Descrição

Pretende-se implementar um sistema de Relógio de Ponto (*Working Time Recorder*) que permita registar a entrada/saída de um colaborador através de um número de identificação de utilizador (*User Identification Number – UIN*) e um código de acesso (*Personal Identification Number - PIN*). O sistema regista a entrada/saída após a inserção correta de um par *UIN* e *PIN*, para além do registo o sistema permite o acesso através de uma porta de acesso.
O sistema de relógio de ponto é constituído por: um teclado de 12 teclas; um ecrã *Liquid Cristal Display* (*LCD*) de duas linhas de 16 caracteres; um mecanismo de abertura e fecho da porta (designado por *Door*) e uma chave de manutenção (designada por *M*) que define se o sistema de relógio de ponto está em modo de Manutenção. O diagrama de blocos do sistema de controlo de acessos é apresentado na Figura 1.

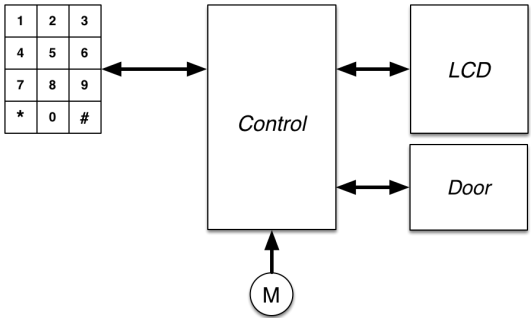


Figura 1 – Sistema de relógio de ponto (*Working Time Recorder*)

Sobre o sistema podem-se realizar as seguintes ações em modo *Registo*:

- **Registo -** Para registar a entrada/saída das instalações, o colaborador deverá inserir os três dígitos correspondentes ao *UIN* seguido da inserção dos quatro dígitos numéricos do *PIN*. Se o par *UIN* e *PIN* estiver correto o sistema apresenta no LCD a hora de entrada e saída, apresentando também o tempo de trabalho acumulado, acionando a abertura da porta. Todas as entradas/saídas deverão ser registadas com a informação de data/hora e *UIN* num ficheiro de registos, designado por *Log File, com* um registo de entrada/saída por linha, com o seguinte formato *"REGIST_DATE;[>](entrada) ou [<](saída);UIN;NAME"*.
- **Alteração do PIN –** Esta ação é realizada se após o processo de autenticação for premida a tecla '#'. O sistema solicita ao utilizador o novo *PIN*, este deverá ser novamente introduzido de modo a ser confirmado. O novo *PIN* só é registado no sistema se as duas inserções forem idênticas.

Nota: A inserção de informação através do teclado tem o seguinte critério: se não for premida nenhuma tecla num intervalo de cinco segundos o comando em curso é abortado; se for premida a tecla '*' e o sistema contiver dígitos limpa todos os dígitos, se não contiver dígitos aborta o comando em curso; quando o dado a introduzir for um campo numérico e for premida uma tecla não numérica, esta deve ser ignorada.

Sobre o sistema podem-se realizar as seguintes ações em modo *Manutenção*:

- **Inserção de utilizador -** Tem como objetivo inserir um novo utilizador no sistema. O sistema atribui o primeiro *UIN* disponível, e espera que seja introduzido pelo gestor do sistema o nome e o *PIN* do utilizador. O nome tem no máximo 16 caracteres, letras maiúsculas e/ou minúsculas e/ou um espaço entre nomes.
- **Remoção de utilizador -** Tem como objetivo remover um utilizador do sistema. O sistema espera que o gestor do sistema introduza o *UIN* e pede confirmação depois de apresentar o nome.
- **Listar os utilizadores –** Permite visualizar no ecrã do PC a lista dos utilizadores presentes nas instalações, apresentando o *UIN,* o nome e a hora de entrada.
- **Desligar –** Permite desligar o sistema de controlo de acessos, este termina após a confirmação do utilizador e a reescrita do ficheiro com a informação dos utilizadores. Esta informação deverá ser armazenada num ficheiro de texto (com um utilizador por linha, com o formato "*UIN;PIN;NAME;ACCUMULATED_TIME;ENTRY_DATE*") que é carregado no início do programa e reescrito no final do programa. O sistema armazena até 1000 utilizadores, que são inseridos e suprimidos através do teclado do PC pelo gestor do sistema.

Nota: Durante a execução das ações em modo manutenção, realizadas através do teclado do PC, não podem ser realizadas ações no teclado do utilizador e no LCD deve constar a mensagem "*Out of Service*".

2 Arquitetura do sistema

O sistema será implementado numa solução híbrida de hardware e software, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por quatro módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD*; *iii*) um módulo de interface e controlo do mecanismo da porta, designado por *Door Controller*; e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em *hardware*, enquanto o módulo de controlo deverá ser implementado em *software* a executar num PC usando linguagem *Kotlin* (ou *Java*).

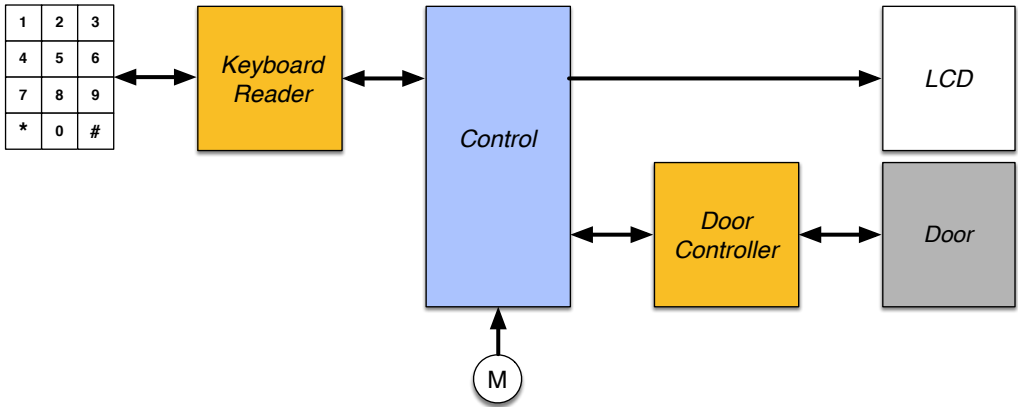


Figura 2 – Arquitetura do sistema que implementa o relógio de ponto (*Working Time Recorder*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o seu código, com quatro bits, ao módulo *Control*. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de três códigos. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, a comunicação entre o módulo *Control* e o módulo *Keyboard Reader* é realizada através de um protocolo série. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD*. O mecanismo da porta, designado por *Door*, é atuado pelo módulo *Control*, através do módulo *Door Controller*.

2.1 Keyboard Reader

O módulo *Keyboard Reader* é constituído por três blocos principais: *i*) o descodificador de teclado (*Key Decode*); *ii*) o bloco de armazenamento (designado por *Key Buffer*); e *iii*) o bloco de entrega ao consumidor (designado por *Key Transmitter*), conforme ilustrado na Figura 3. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

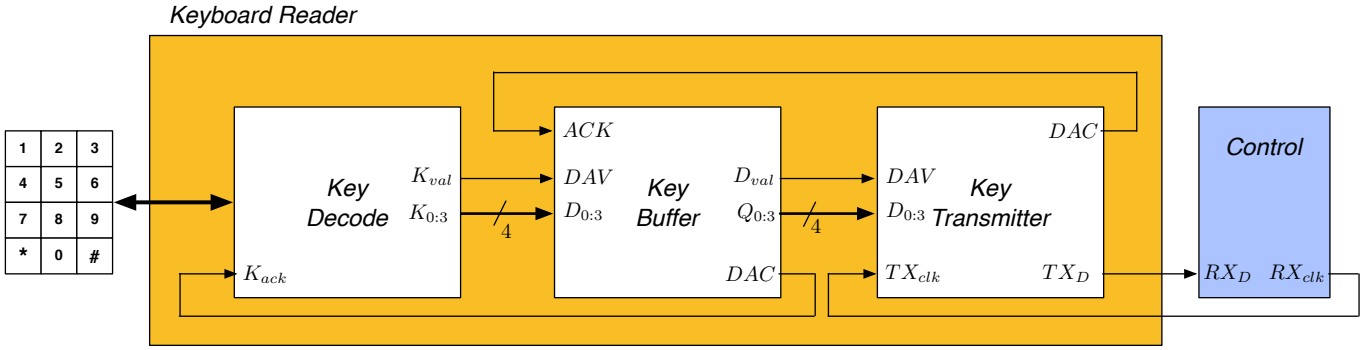
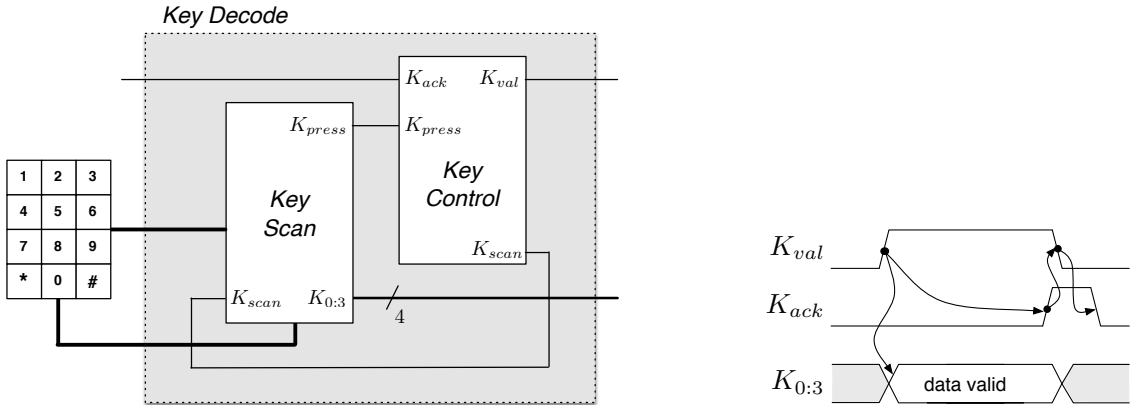


Figura 3 – Diagrama de blocos do módulo *Keyboard Reader*

2.1.1 Key Decode

O bloco *Key Decode* deverá implementar um descodificador de um teclado matricial 4x3 por *hardware*, sendo constituído por três sub-blocos: *i)* um teclado matricial de 4x3; *ii)* o bloco *Key Scan,* responsável pelo varrimento do teclado; e *iii)* o bloco *Key Control*, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura 4a.

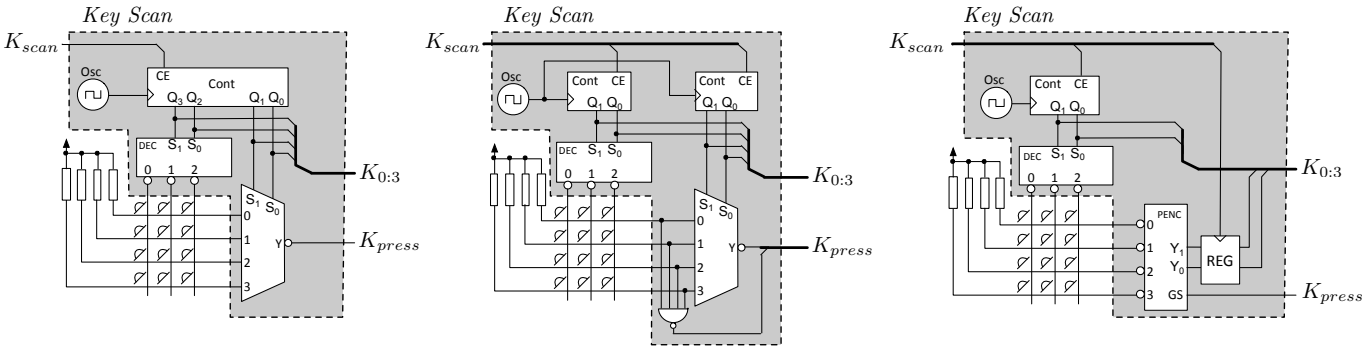
O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key Buffer*), define que o sinal K_{val} é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal K_{ack} for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 4b.



a) Diagrama de blocos b) Diagrama temporal

Figura 4 – Bloco *Key Decode*

O bloco *Key Scan* deverá ser implementado de acordo com um dos diagramas de blocos representados na Figura 5, enquanto o desenvolvimento e a implementação do bloco *Key Control* ficam como objeto de análise e estudo, devendo a sua arquitetura ser proposta pelos alunos.



a) versão I b) versão II c) versão III

Figura 5 - Diagrama de blocos do bloco *Key Scan*

2.1.2 Key Buffer

O bloco *Key Buffer* a desenvolver corresponderá a uma estrutura de armazenamento de dados, com capacidade para armazenar uma palavra de quatro bits. A escrita de dados no bloco *Key Buffer*, cujo diagrama de blocos é apresentado na Figura 6, inicia-se com a ativação do sinal *DAV* (*Data Available*) pelo sistema produtor, neste caso pelo bloco *Key Decode*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o bloco *Key Buffer* regista os dados $D_{0:3}$ em memória. Concluída a escrita em memória, ativa o sinal *DAC* (*Data Accepted*) para informar o sistema produtor que os

dados foram aceites. O sistema produtor mantém o sinal *DAV* ativo até que o sinal *DAC* seja ativado. O bloco *Key Buffer* só desativa o sinal *DAC* após o sinal *DAV* ter sido desativado.
A implementação do bloco *Key Buffer* deverá ser baseada numa máquina de controlo (*Key Buffer Control*) e num registo (*Output Register*).

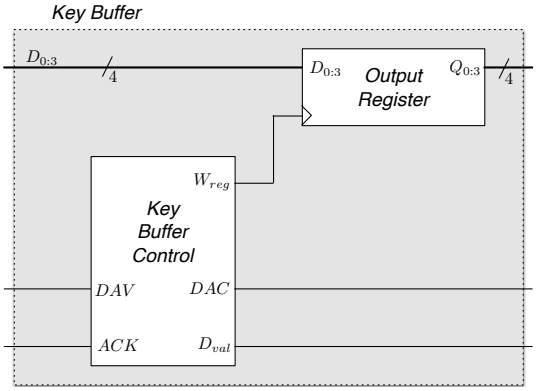


Figura 6 – Diagrama de blocos do bloco *Key Buffer*

O sub-bloco *Key Buffer Control* do bloco *Key Buffer* também é responsável pela interação com o sistema consumidor, neste caso o bloco *Key Transmitter*. Quando pretende ler dados do bloco *Key Buffer*, o bloco *Key Transmitter* aguarda que o sinal D_{val} fique ativo, recolhe os dados e ativa o sinal *ACK* para indicar que estes já foram consumidos. Logo que o sinal *ACK* fique ativo, o módulo *Key Buffer Control* deve invalidar os dados baixando o sinal D_{val}. Para que uma nova palavra possa ser armazenada será necessário que o bloco *Key Transmitter* tenha desativado o sinal *ACK*.

2.1.3 Key Transmitter

O bloco *Key Transmitter* a desenvolver corresponderá a uma estrutura de transmissão série, com capacidade para armazenar e transmitir uma palavra de quatro bits. A escrita de dados no bloco *Key Transmitter* inicia-se com a ativação do sinal *DAV* (*Data Available*) pelo sistema produtor, neste caso pelo bloco *Key Buffer*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o bloco *Key Transmitter* regista os dados $D_{0:3}$ na sua memória interna. Concluída a escrita na memória interna, ativa o sinal *DAC* (*Data Accepted*) para informar o sistema produtor que os dados foram aceites. O sistema produtor mantém o sinal *DAV* ativo até que o sinal *DAC* seja ativado. O bloco *Key Transmitter* só desativa o sinal *DAC* após o sinal *DAV* ter sido desativado.
O bloco *Key Transmitter* também é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*.
Este bloco quando tem dados para transmitir, sinaliza o módulo *Control* através duma transição descendente do sinal TX_D, enquanto o sinal TX_{clk} está no valor lógico zero. A cada transição ascendente do sinal TX_{clk} o bloco *Key Transmitter* coloca o sinal TX_D com o valor lógico necessário em cada instante, de acordo com o protocolo representado na Figura 7. Para que uma nova palavra possa ser armazenada neste bloco será necessário que o módulo *Control* tenha recebido todos os bits. A implementação do bloco *Key Transmitter* fica como objeto de análise e estudo, devendo a sua arquitetura ser proposta pelos alunos.

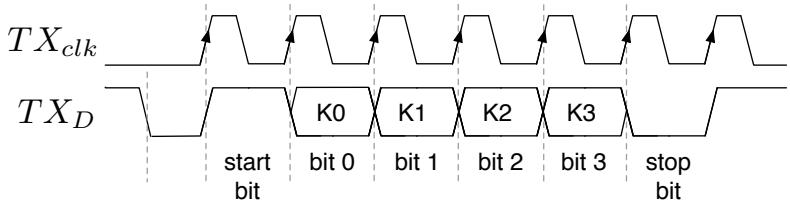


Figura 7 – Protocolo de comunicação com o módulo *Key Transmitter*

2.2 Interface com o *LCD*

O módulo de interface com o *LCD* implementa a interface entre o *Control* e o *LCD*, numa comunicação síncrona paralela a 4 bit, conforme representado na Figura 8.

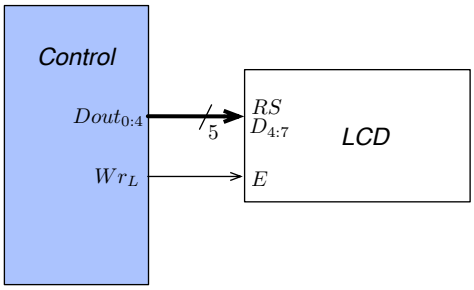


Figura 8 – Diagrama de blocos do módulo de interface com o *LCD*

2.3 Door Controller

O módulo de controlo e interface com o mecanismo da porta (*Door Controller*), recebe em paralelo a informação enviada pelo *Control*, conforme representado na Figura 9.
O bloco *Door* Controller após ter recebido um comando válido entregue pelo *Control*, deverá proceder à atuação deste no mecanismo da porta, ativando de imediato o sinal *busy*. Se o comando recebido for de abertura da porta, o *Door Controller* deverá colocar o sinal $OnnOff$ e o sinal $OpennClose$ no valor lógico '1', até o sensor de porta aberta (FC_{open}) ficar ativo. No entanto se o comando for de fecho, o *Door Controller* deverá ativar o sinal $OnnOff$ e colocar o sinal $OpennClose$ no valor lógico '0', até o sensor de porta fechada (FC_{close}) ficar ativo. Se durante o fecho for detetada uma pessoa na zona da porta, através do sensor de presença (P_{detect}), o sistema deverá interromper o fecho reabrindo a porta. Após a interrupção do fecho da porta o bloco *Door Controller* deverá permitir de forma automática, ou seja, sem necessidade de envio de um novo comando, o encerramento da porta e o finalizar do comando de fecho. Após concluir qualquer um dos comandos o *Door Controller* sinaliza o *Control* que está pronto para processar um novo comando através da desativação do sinal *busy*.

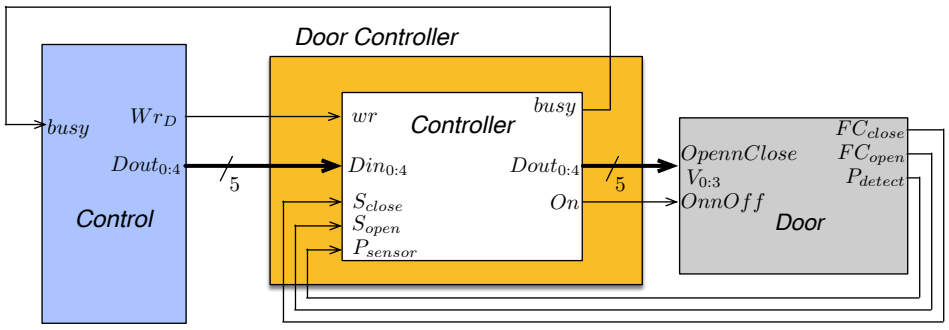


Figura 9 – Diagrama de blocos do módulo *Door Controller*

2.4 *Control*

A implementação do módulo *Control* deverá ser realizada em *software*, usando a linguagem *Kotlin* (ou *Java*) e seguindo a arquitetura lógica apresentada na Figura 10.

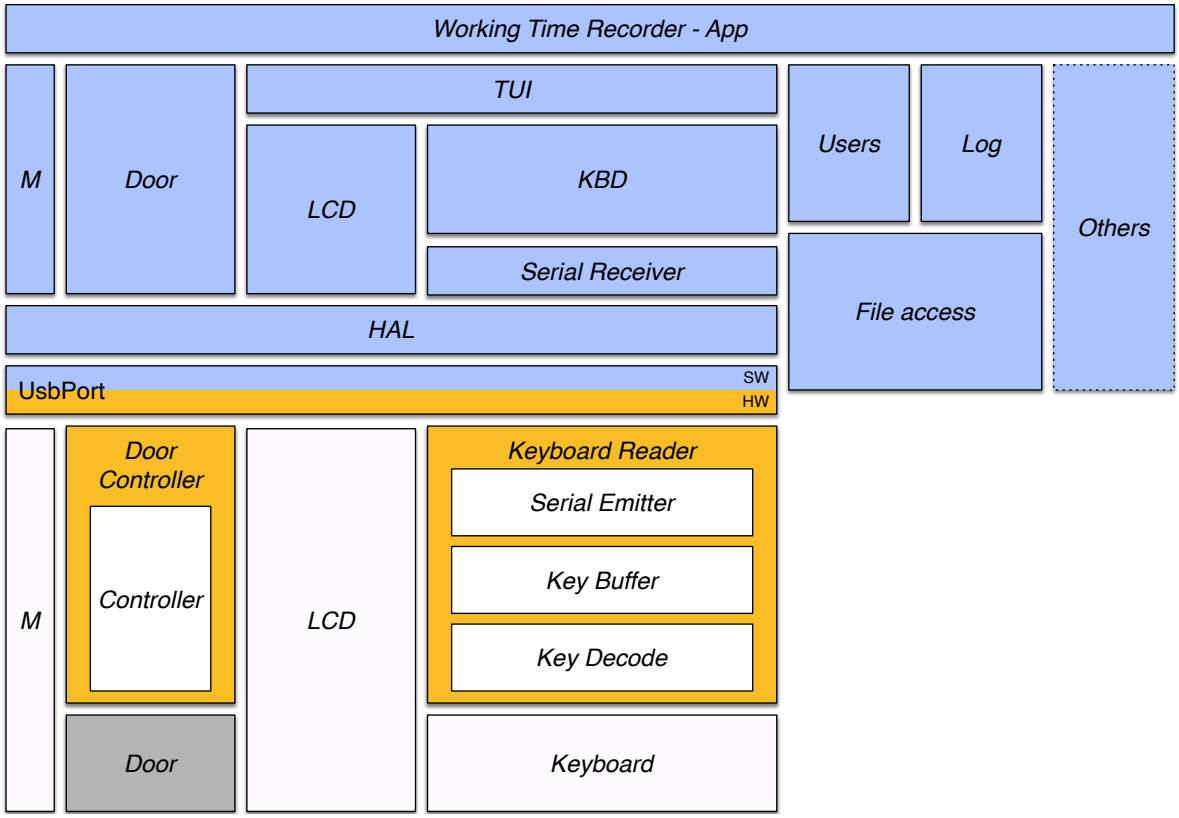


Figura 10 – Diagrama lógico do Relógio de Ponto (*Working Time Recorder*)

As assinaturas das principais classes a desenvolver são apresentadas nas próximas secções, respetivamente em *Kotlin* (2.4.1) e *Java* (2.4.12.4.2). As restantes são objeto de análise e decisão livre.

2.4.1 Linguagem Kotlin

2.4.1.1 HAL

```
object HAL {   // Virtualiza o acesso ao sistema UsbPort
   // Inicia a classe
   fun init() ...
   // Retorna true se o bit tiver o valor lógico ‘1’
   fun isBit(mask: Int): Boolean ...
   // Retorna os valores dos bits representados por mask presentes no UsbPort
   fun readBits(mask: Int): Int ...
   // Escreve nos bits representados por mask o valor de value
   fun writeBits(mask: Int, value: Int) ...
   // Coloca os bits representados por mask no valor lógico ‘1’
   fun setBits(int mask) ...
   // Coloca os bits representados por mask no valor lógico ‘0’
   fun clrBits(mask: Int) ...
}
```

2.4.1.2 KBD

```
object KBD {   // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   const val NONE = 0;
   // Inicia a classe
   fun init() ...
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
   fun getKey(): Char ...
   // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
   fun waitKey(timeout: Long): Char ...
}
```

2.4.1.3 Classe KeyReceiver

```
object KeyReceiver {  // Recebe trama do Keyboard Reader.
   // Inicia a classe
   fun init() ...
   // Recebe uma trama e retorna o código de uma tecla caso exista
   fun rcv(): Int ...
}
```

2.4.1.4 Classe LCD

```
object LCD {   // Escreve no LCD usando a interface a 4 bits.
   private const val LINES = 2, COLS = 16; // Dimensão do display.
   // Escreve um nibble de comando/dados no LCD
   private fun writeNibble(rs: Boolean, data: Int) ...
   // Escreve um byte de comando/dados no LCD
   private fun writeByte(rs: Boolean, data: Int) ...
   // Escreve um comando no LCD
   private fun writeCMD(data: Int) ...
   // Escreve um dado no LCD
   private fun writeDATA(data: Int) ...
   // Envia a sequência de iniciação para comunicação a 4 bits.
   fun init() ...
   // Escreve um caráter na posição corrente.
   fun write(c: Char) ...
   // Escreve uma string na posição corrente.
   fun write(text: String) ...
   // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
   fun cursor(line: Int, column: Int) ...
   // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
   fun clear() ...
}
```

2.4.1.5 Classe Door

```
object Door{          // Controla o mecanismo da porta.
   // Inicia a classe, estabelecendo os valores iniciais.
   fun init() ...
   // Envia comando para abrir a porta, indicando a velocidade
   fun open(speed: Int) ...
   // Envia comando para fechar a porta, indicando a velocidade
   fun close(speed: Int) ...
   // Retorna true se o tiver terminado o comando
   fun isFinished(): Boolean ...
}
```

2.4.2 Linguagem Java

2.4.2.1 Classe HAL

```
public class HAL {    // Virtualiza o acesso ao sistema UsbPort
   // Inicia a classe
   public static void init() ...
   // Retorna true se o bit tiver o valor lógico '1'
   public static boolean isBit(int mask) ...
   // Retorna os valores dos bits representados por mask presentes no UsbPort
   public static int readBits(int mask) ...
   // Escreve nos bits representados por mask o valor de value
   public static void writeBits(int mask, int value) ...
   // Coloca os bits representados por mask no valor lógico '1'
   public static void setBits(int mask) ...
   // Coloca os bits representados por mask no valor lógico '0'
   public static void clrBits(int mask) ...
}
```

2.4.2.2 Classe KBD

```
public class KBD {    // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   public static final char NONE = 0;
   // Inicia a classe
   public static void init() ...
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
   public static char getKey() ...
   // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
   public static char waitKey(long timeout) ...
}
```

2.4.2.3 Classe KeyReceiver

```
public class KeyReceiver {   // Recebe trama do Keyboard Reader.
   // Inicia a classe
   public static void init() ...
   // Recebe uma trama e retorna o código de uma tecla caso exista
   public static int rcv() ...
}
```

2.4.2.4 Classe LCD

```
public class LCD {    // Escreve no LCD usando a interface a 4 bits.
   private static final int LINES = 2, COLS = 16; // Dimensão do display.
   // Escreve um nibble de comando/dados no LCD
   private static void writeNibble(boolean rs, int data) ...
   // Escreve um byte de comando/dados no LCD
   private static void writeByte(boolean rs, int data) ...
   // Escreve um comando no LCD
   private static void writeCMD(int data) ...
   // Escreve um dado no LCD
   private static void writeDATA(int data) ...
   // Envia a sequência de iniciação para comunicação a 4 bits.
   public static void init() ...
   // Escreve um caráter na posição corrente.
   public static void write(char c) ...
   // Escreve uma string na posição corrente.
   public static void write(String txt) ...
   // Envia comando para posicionar cursor ('lin':0..LINES-1 , 'col':0..COLS-1)
```

```
    public static void cursor(int lin, int col) ...
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    public static void clear() ...
}
```

2.4.2.5 Classe Door

```
public class Door{            // Controla o mecanismo da porta.
    // Inicia a classe, estabelecendo os valores iniciais.
    public static void init() ...
    // Envia comando para abrir a porta, indicando a velocidade
    public static void open(int speed) ...
    // Envia comando para fechar a porta, indicando a velocidade
    public static void close(int speed) ...
    // Retorna true se o tiver terminado o comando
    public static boolenan isFinished() ...
}
```

3 Calendarização do projeto

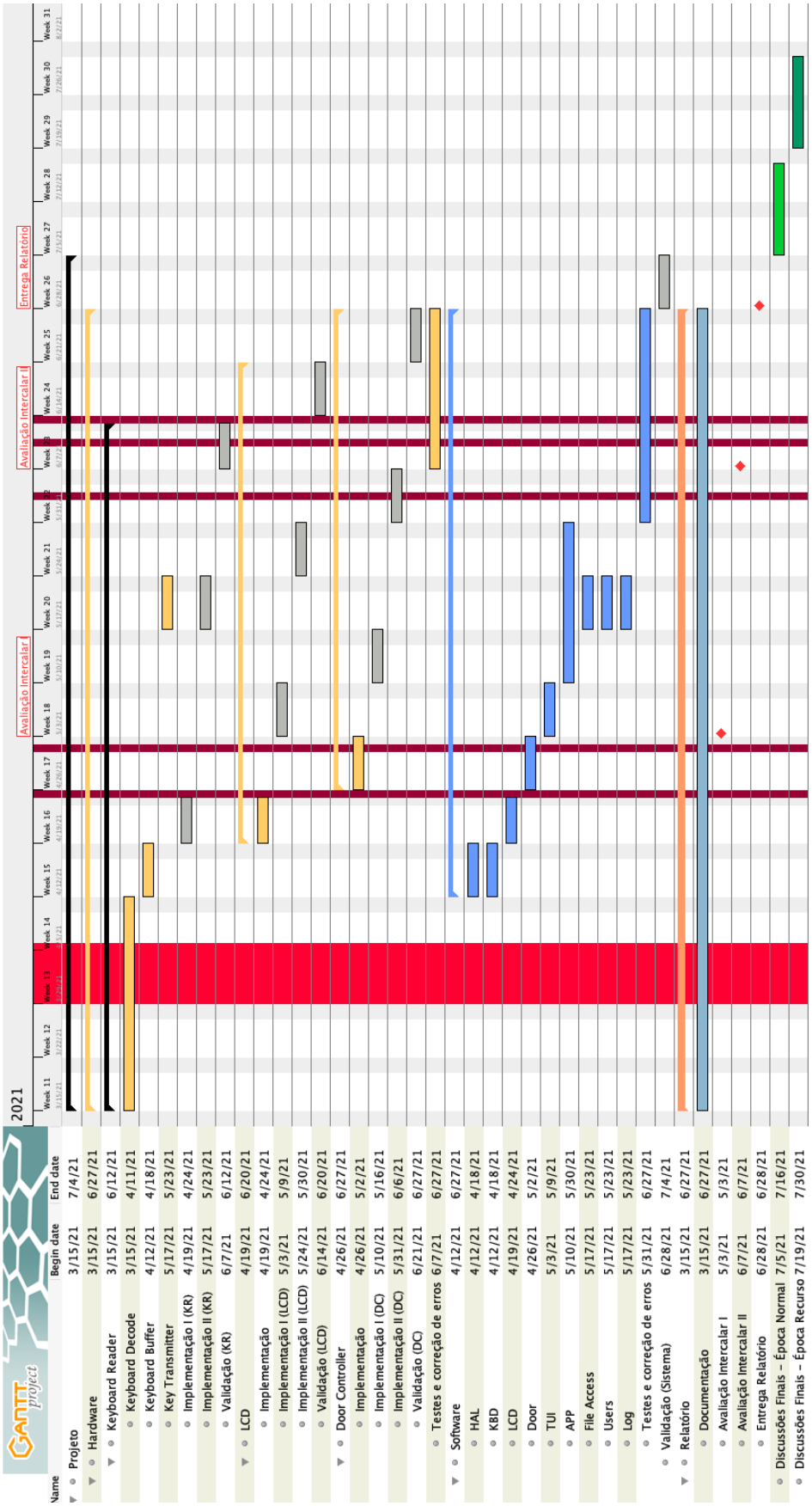


Figura 11 – Diagrama de *Gantt* relativo à calendarização do projeto