MATCH!

Ingenieursproject: Infrarood communicatie

Wannes Baes, Vincent Frippiat

Begeleiders: Francis Wyffels, Tom Neutens, Thomas Lips ,Victor-Louis de Gusseme

Academiejaar 2021-2022

Inhoudstafel

Tabel met specificaties

Parameter	Waarde
Kloksnelheid van de microcontroller	16 MHz
Pin HIGH	5V
Pin LOW	0V
Frequentie van infraroodlicht	300 GHz – 400 THz
Golflengte van infraroodlicht	$700\ nm - 1\ mm$
Frequentie van het infraroodsignaal	$38\ kHz$
Periode van het infraroodsignaal	$26.32\ \mu s$
Aantal infrarood pulsen bij het verzenden van een 1-bit	120
Aantal infrarood pulsen bij het verzenden van een 0-bit	40
Duur van het verzenden van een 1-bit.	2.884 ms
Duur van het verzenden van een 1-bit.	0.960 ms
Prescaler Timer1	256
Compare Value Timer1	5
Bitpatron positie LEFT in GAME_PLAYER	00
Bitpatron positie UP in GAME_PLAYER	10
Bitpatron positie RIGHT in GAME_PLAYER	01
Aantal bits per bericht	6

1. INLEIDING

Gedurende afgelopen semester werkten we aan een multiplayer spel genaamd ‘MATCH!’. Het spel wordt gespeeld door 2 spelers, elk op hun eigen dwenguino-bordje. We hebben dit spel geprogrammeerd in de programmeertaal C. In het spel is het de bedoeling te gokken welke ‘avatar’ de tegenstander heeft gekozen.

De werking van het spel berust op infrarood communicatie tussen twee microcontrollers. In de eerste twee delen van het verslag zal dit in detail worden besproken. Daarna zal op de werking van het spel zelf worden ingegaan. Vervolgens overlopen we het verloop van het project en tot slot gaan over tot een conclusie.

2. BOUWSTENEN VAN INFRAROOD COMMUNICATIE

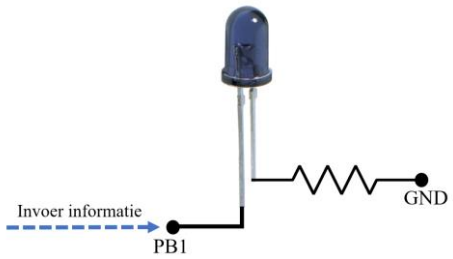
2.1. *Inleiding*

Om het spel MATCH! te kunnen spelen moeten de twee microcontrollers om de beurt informatie naar elkaar kunnen sturen. Dit gebeurt met behulp van communicatie tussen de infrarood LED, die aangesloten is op een microcontroller en die het signaal verstuurt, en de infrarood sensor, die aangesloten is op een andere microcontroller en die het signaal ontvangt.

Elk lichaam dat warmte bevat zendt infraroodlicht uit. Opdat de sensor hier niet op zou reageren is deze zodanig ingesteld dat enkel infraroodlicht op een frequentie van 38 KHz wordt ontvangen.

2.2. *IR-led*

De IR-led is met zijn korte been in serie verbonden aan een weerstand en vervolgens met de ground GND. Het lange been van de led is verbonden aan de pin PB1 van de microcontroller. Deze led zendt IR-licht uit zodra de pin PB1 als output aangesteld wordt en de waarde van deze pin HIGH zet (zie figuur 1). We kozen pin PB1 omdat deze pin nergens anders in gebruik is. Zo vermijden we eventuele overschrijvingen en kunnen we de pin standaard instellen als output. Deze pin staat niet in verbinding met een interrupt, wat ook niet nodig is.

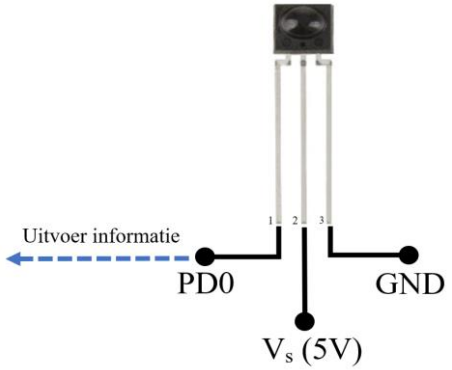


Figuur 1: Schematische verbinding van de IR-led aan de microcontroller.

2.3. *IR sensor*

De IR-sensor is een sensor waarvan de toestand in rust op HIGH staat, en op LOW van zodra een signaal ontvangen wordt. De sensor mag geen signaal voor een te lange tijdsperiode ontvangen. Indien dit toch het geval is zal de sensor vanzelf weer op HIGH schakelen vanwege de beperkte duty cycle.

De IR-sensor heeft drie verschillende poten, zoals te zien is op figuur 2. De tweede en derde poot zijn respectievelijk verbonden aan de supply voltage pin V_S (5V) en de ground GND. De receptie van infrarood signalen gebeurt onder andere met behulp van interrupt service routines (software proces aangeroepen door een interruptverzoek van een hardware apparaat). Aangezien de receptie van een infrarood signaal een delicaat proces is, mag deze ISR niet onderbroken worden door andere interrupts. Om deze reden hebben we gekozen om de eerste poot van de IR ontvanger aan de pin PD0 van de microcontroller te verbinden (zie poot 1 op figuur 2): deze pin kan de externe ISR met de hoogste prioriteit (aan het adres INT0) oproepen. Daarnaast overlapt de pin PD0 met geen enkele andere functie van het dwenguino-bord die we willen gebruiken. We stellen deze pin dus standaard in als input.



Figuur 2: Schematische verbinding van de IR-led aan de microcontroller.

De ontvangst van signalen op de infrarood-sensor hebben we in de eerste plaats geverifieerd met de logic Analyzer en het daaraan geassocieerde programma PulseView. In een periode die we zelf bepalen toont dit programma ons wanneer de IR-led informatie uitzendt en wanneer de IR-sensor deze ontvangt.

3. INFRAROOD COMMUNICATIE: PROTOCOL

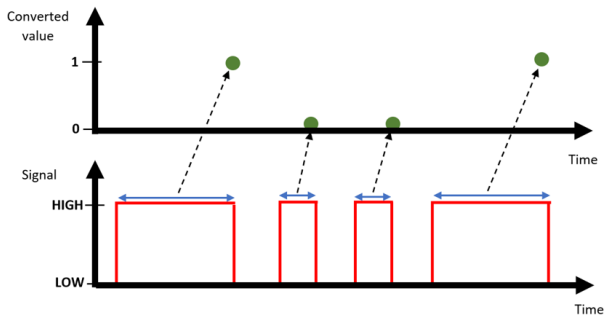
3.1. *Inleiding*

Voor een geldige communicatie tussen de verzender en de ontvanger van een bericht is een communicatieprotocol elementair. In deze paragraaf wordt de implementatie van dit protocol besproken. De algemene structuur en de pulsatiebreedte modulatie komen uitgebreid aan bod. Voor een optimaal begrip verwijzen we graag naar figuur 3 in de bijlage, die een algemeen overzicht geeft.

3.2. *Verzending van signalen*

De gebruikte infrarood receiver ontvangt enkel infrarood signalen aan een specifieke frequentie (38 kHz). Om een functionele communicatie te hebben, is het dus cruciaal om het infrarood licht met deze frequentie te verzenden via de led: hierdoor flikkert de led met een totale periode $T = \frac{1}{38kHz} = 26.32\,\mu s$. Met andere woorden, de LED gaat telkens aan en uit voor ongeveer $T_{aan} = T_{uit} = \frac{1}{38kHz} * \frac{1}{2} = 13.16\,\mu s$. De gebruikte methode voor de emissie van dit signaal is eenvoudig maar efficiënt: met behulp van de functie $\_delay\_\mu s()$ wordt er telkens voor een duur $T_{uit} = T_{aan}$ gewacht, om daarna de LED om te schakelen. Deze methode is mogelijk omdat het aantal instructies in verband met de infrarood communicatie gelimiteerd is. Het gebruik van bijvoorbeeld timer-interrupts zou in ons geval over engineering en dus overbodig zijn.

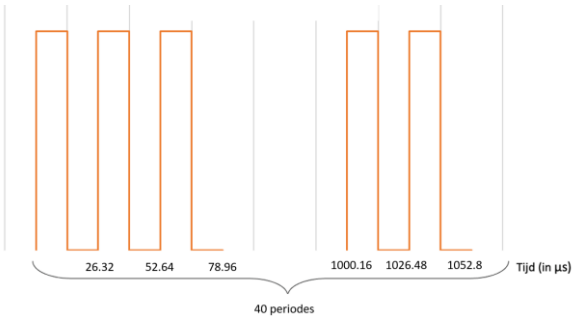
Voor het versturen van bits wordt gebruik gemaakt van de pulsatiebreedte modulatie (afgekort PWM). Dit is een modulatietechniek waarbij pulsen worden uitgezonden aan een variërende breedte, waarbij elke breedte een specifieke waarde representeert. Algemeen geldt, hoe groter de waarde, hoe langer het signaal. Aangezien wij enkel geïnteresseerd zijn in het versturen van bits worden er slecht 2 verschillende signalen verstuurd.



Figuur 4: Grafische representatie van de PWM-methode (beneden: verzonden signaal, boven: ontvangen en verwerkt signaal).

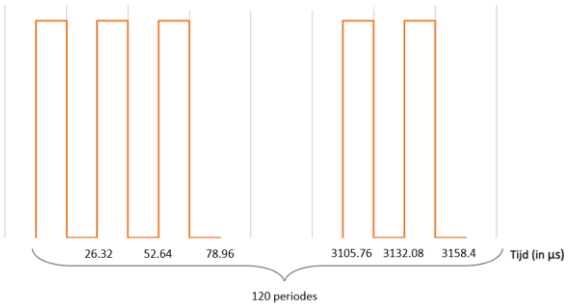
Er wordt gelet op het feit dat de receiver een signaal niet mag ontvangen voor een te lange tijdsduur. Bovendien willen we dat de communicatie zo vlot en snel mogelijk verloopt, dus te lange signalen zijn inefficiënt. Daarentegen willen we fouten bij het decoderen van de signalen tegengaan door het verschil tussen een signaal dat een 0 en een 1 voorstelt groot genoeg te kiezen. Te korte signalen zijn dus ook onbruikbaar. We zijn dus op zoek gegaan naar het evenwicht tussen deze twee drijfveren. We bekwamen:

- 0-bit: gedurende 40 periodes T van 26.32 μs een IR signaal doorsturen. Dit patroon is op de grafiek hieronder (figuur 5) weergegeven.



Figuur 5: Grafische representatie van de emissie van een 0-bit aan een frequentie van 38 kHz.

- 1-bit: gedurende 120 periodes T van 26.32 μs een IR signaal doorsturen. Dit patroon is op de grafiek hieronder (figuur 6) weergegeven.



Figuur 6: Grafische representatie van de emissie van een 1-bit aan een frequentie van 38 kHz.

Wanneer de infrarood LED een bericht uitzendt, wordt interrupt nul (INT0), waaraan de sensor verbonden is, uitgeschakeld om te vermijden dat de microcontroller het verzonden bericht zelf verwerkt.

De functie $sendSignal$ wordt gebruikt voor het versturen van individuele bits (zie figuur 3). Voor het verzenden van een volledige bitsequentie wordt $sendSignal$ opgeroepen via de functie $sendMessage$ voor elk element in $messageToSend$. De bit sequentie wordt met een opwaartse bit nummering verzonden.

3.3. *Ontvangst van signalen*

Bij ontvangst wordt er gemeten of er een 0 of 1 is doorgestuurd in functie van hoelang de input LOW blijft (dit is dus de breedte van het signaal). Om de breedte van de ontvangen signalen te bepalen, wordt de ISR aan het adres INT0 ingesteld als een interrupt on changing edge: deze interrupt wordt opgeroepen zodra een verandering optreedt in de output van de receiver. Vervolgens wordt gebruik gemaakt van de variabele $receivingSignal$. Deze wordt op 0 gezet bij een rising edge (geen signaal meer ontvangen) en op 1 gezet bij een falling edge (nieuw signaal ontvangen) in de output van de receiver. Zodra $receivingSignal$ gelijk is aan 1, stijgt de waarde van de variabele $lengthSignal$ incrementeel: de variabele geeft aan "hoe breed" het signaal is (zie figuur 3).

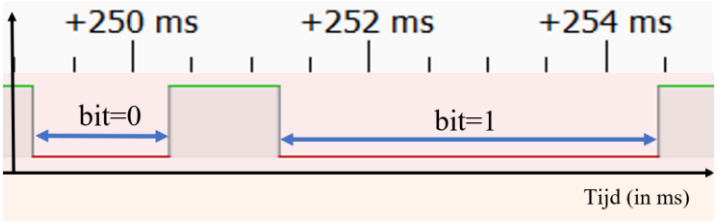
Deze optelling gebeurt met behulp van de timer 1. Door de prescaler van deze timer gelijk te stellen aan 256 bekomen we een klokfrequentie van 62500 tikken per seconde. De compare value wordt ingesteld op 5, zodat om de 12 500 keer per seconde een timer interrupt routine wordt opgeroepen. Deze waarde mag niet te laag zijn, want dan worden er onnodig veel interrupt routines uitgevoerd en de code dus te hard vertraagd. Daarentegen mag de waarde ook niet te hoog zijn, want dan wordt het verschil tussen een signaal van een 1-bit of 0-bit slecht opgemeten.

Zoals de figuur 3 aangeeft, stopt deze optelling bij een rising edge van de pin van de receiver als, $receivingSignal$ gelijk aan 1 is. Vervolgens wordt de waarde van $lengthSignal$ gekopieerd en doorgegeven in de functie $defineBit$. Hierna wordt $lengthSignal$ gelijk aan 0 gesteld (voor de ontvangst van het volgende signaal).

In $defineBit$ geeft de variabele $lengthSignal$ aan hoe breed het ontvangen signaal is. De waarde van *lengthSignal* dat overeenkomt met een 1-signaal is 36 en met een 0-signaal 12. Dit kan berekend worden door de duur van het signaal te delen door het aantal timer-interrupts dat in die duur worden gegenereerd. De waarde die overeenkomt met een 1-bit is dus 3 keer groter dan die dat overeenkomt met een 0-bit. Dit verschil is ruim genoeg en toch geen overbodige precisie.

Aangezien de hardware en software hun taken niet bij elke ontvangst op exact dezelfde wijze executeren, is het vanzelfsprekend dat $lengthSignal$ zelden exact gelijk zal zijn aan één van deze waarden. Desalniettemin zal de waarde $lengthSignal$ er dichtbij zijn. Hierdoor is dit de formele definitie van een ontvangen 0 en 1:

- Als $\text{lengthSignal} < \frac{\text{LENGTH\_SIGNAL\_ZERO+LENGTH\_SIGNAL\_ONE}}{2}$: de ontvangen bit is 0
- Als $\text{lengthSignal} \geq \frac{\text{LENGTH\_SIGNAL\_ZERO+LENGTH\_SIGNAL\_ONE}}{2}$: de ontvangen bit is 1



Figuur 7: Grafiek van de ontvangst van de bits 0 en 1.

Het ontvangen van bit sequenties verschilt niet zo veel van het ontvangen van een bit. Hierbij wordt er eerst een array $messageReceived$ aangemaakt om de binaire waarden op te slaan. Vervolgens wordt er gebruik gemaakt van een integer, als index voor de array, die incrementeel stijgt zodra een nieuwe waarde is toegevoegd aan de array en terug op nul wordt gezet wanneer de volledige boodschap is ontvangen.

3.4. *Repetitebits*

Infrarood communicatie is een zeer betrouwbare communicatiemethode. Toch kunnen er soms fouten voorkomen. Om de kans op deze fouten te verkleinen bevatten de berichten in MATCH! repetitie bits: dit is een herhaling van het initieel patroon in het doorgestuurde bericht.

In MATCH! worden alleen maar de waarde 0, 1 en 2 gestuurd via infraroodcommunicatie. Dit zijn dus waarden met maximaal 2 bits in een binaire voorstelling. In MATCH! bevat het bericht driemaal deze initiële patroon (in totaal zes bits). Wanneer het bericht wordt ontvangen, verifieert de functie $checkRepetitionBit$ voor elke positie k in het initiële bericht (dus $k = 0$ of $k = 1$) welke bit het meeste voorkomt in de verzameling $\{messageReceived[k], messageReceived[k+2], messageReceived[k+4]\}$. Er gaan nooit twee bits evenveel voorkomen omdat deze verzameling een oneven aantal elementen bevat.

Figuur 8 (zie bijlage) is een voorbeeld van deze methode: het ontvangen bericht is 10 omdat de meest voorkomende bit op positie 0 de waarde 1 is en de meest voorkomende bit op positie 1 de waarde 0 is.

Deze methode dedecteert dus niet alleen de eventuele fout, maar corrigeert deze ook. Ze verwijdert uiteraard niet elke mogelijke fout, maar reduceert toch aanzienlijk de kans op imperfecties in het ontvangen bericht. Als we meer repetitie bits gebruiken zal de kans op fouten nog sterker dalen. Toch hebben we dit niet gedaan omdat het bericht niet te lang mag zijn. Het versturen van signalen zou dan onnodig veel tijd innemen.

4. MATCH!: WERKING

In dit onderdeel wordt er uitgebreide uitleg gegeven over de werking en implementatie van MATCH! De architectuur van het spel berust op een verdeling van het spel in verschillende scenario's. We noemen deze scenarios (in het Engels) de game states. De code van dit spel is geïmplementeerd op basis van één functie (process_game) die in functie van de meegegeven game state verschillende taken zal uitvoeren. In figuur 9 (zie bijlage) hieronder is de werking van het spel (multiplayer-mode) schematisch gerepresenteerd.

4.1. *Startmenu*

Wanneer het spel wordt geladen bij twee spelers is hun eerste game state MENU_INTRO. Eén van de twee spelers kan een spel opstarten door op de WEST- , EAST- of NORTH-knop te drukken, die respectievelijk zijn verbonden aan de pinnen PE4, PE6, en PE7. Voor het detecteren van input aan deze knoppen wordt gebruik gemaakt van polling. Er werd niet gekozen voor het gebruik van interrupts komend van de knoppen omdat ze interfereerden met de verzending en de ontvangst van het signaal. Voor de komende uitleg noemen we X de speler die een spel start (een INITIATOR) en Y de speler die een verzoek (om te spelen) van X krijgt (een RECEIVER).

4.2. *Keuze spelsoort*

Nadat X het spel start is zijn of haar spel in de game state GAME_MENU_TYPE: hier kan hij of zij kiezen tussen spelen tegen de andere persoon of tegen de AI (equivalent aan de andere modus, maar er wordt direct een 'antwoord' van de tegenstander gegeneerd). Deze tweede optie is echter een additie (niet van centraal belang) aan het spel en redelijk analoog aan de eerste modus: enkel de multiplayer-modus zal in detail worden besproken. Als X dus kiest voor een spel tegen Y komt het spel van X terecht in de game state GAME_PLAYER.

4.3. *Selectie positie*

Bij de game state GAME_PLAYER kan de speler kiezen tussen één van de drie voorgestelde posities door te drukken op de EAST-, NORTH- of WEST-knop om respectievelijk de linkse, middelste of rechtse positie te kiezen. Zodra dit gedaan is, verzendt de microcontroller van X een zes-bit signaal naar de microcontroller van Y (zie figuur 9). Dit bericht bevat uiteraard de keuze van X: WEST is gepresenteerd door de bitsequentie '00', NORTH DOOR '10' en EAST door '01' . Vervolgens krijgt het spel van X een nieuwe game state: GAME_WAITING (zie figuur 9).

Het is in de code vastgesteld dat een speler alleen ontvangen berichten mag verwerken als Y in de game modes MENU_INTRO of GAME_WAITING: op die manier kan een signaal van X geen spel onderbreken bij zijn tegenstander Y (bijvoorbeeld tegen de AI of tegen een speler op een andere microcontroller).

4.4. *Receptie en antwoord van de tegenstander*

Speler Y, waarbij de game state nog MENU_INTRO zou moeten zijn indien hij of zij zelf geen spel is begonnen, ontvangt het zes-bit signaal. Deze ontvangst gebeurt met behulp van het protocol dat in deel 3.3. is besproken. Hierdoor wordt de nieuwe game state bij Y direct GAME_PLAYER (zie figuur 9). Analoog aan de werking van GAME_PLAYER bij X moet Y een positie kiezen, waarna Y een bericht (met zijn of haar eigen keuze) verstuurt naar X.

4.5. *Onthulling van de winnaar*

Na het versturen van het bericht berekent de microcontroller van Y of de twee posities matchen of niet. Afhankelijk van de uitkomst schakelt het spel om in een game state GAME_VICTORY of GAME_DEFEAT. Zodra de microcontroller van X per infraroodcommunicatie de positie van Y heeft ontvangen, volgt hij hetzelfde proces als de microcontroller van Y. Als Y hetzelfde als X heeft gekozen, wint X.

In het tegengestelde geval wint Y.

5. VERLOOP

5.1. *Initiatie*

Het ingenieursproject is begonnen met een kennismaking met het beschikbaar materieel: de microcontroller maar ook de infrarood led en receiver. Gedurende deze periode werden we geconfronteerd met de nadelen van het werken met hardware: tijdens de programmatie wordt er veel vaker rekening gehouden met fysische problemen van de componenten (zoals bijvoorbeeld het gebruik van de button bounce bij de knoppen).

5.2. *Infrarood communicatie*

We zijn begonnen met testen op de infrarood led, waarbij we vroeg de term overengineering hebben leren kennen. Een goed voorbeeld hiervan is het uitzenden van een signaal met de infrarood led: we hebben gedurende een volledige sessie geprobeerd om een signaal via deze led te verzenden met behulp van een methode die gebruikt maakt van timer interrupt, om uiteindelijk de volgende sessie te ontdekken dat het efficiënter zou zijn om dit simpelweg met delays te doen.

Bij het bepalen en de implementatie van het communicatie protocol werden we snel geconfronteerd met onze chaotische organisatie. Dit heeft veel tijd gekost, maar doorheen het semester hebben we geleidelijk geleerd dat een strikte verdeling van de taken en een duidelijk stappenplan elementair zijn. We hebben hierbij bovendien

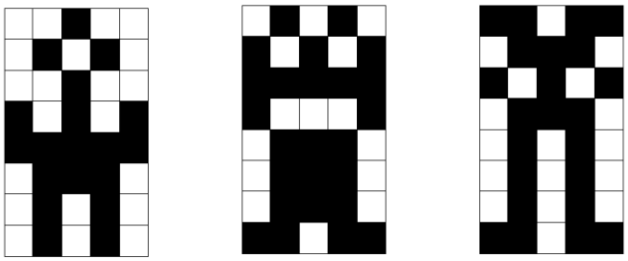
geleerd dat technische problemen bij projecten van deze soort onvermijdelijk zijn. Ze moeten altijd in achting worden genomen bij het opstellen van een planning. We hebben zo bijvoorbeeld de infrarood-receptie voor drie weken moeten uitstellen omdat één van onze laptops zich niet meer correct kon verbinden met de Dwengo-boards.

5.3. *Implementatie van MATCH!*

Na het vaststellen van onze infrarood protocol hebben we wegens tijdstekort besloten om ons eerste idee (een spel vergelijkbaar met een zeeslag waarbij de tegenstander kan bewegen) om te vormen naar een simpeler concept: MATCH! . We hebben snel ontdekt dat de opgedane ervaring (van dit semester) ons veel heeft geholpen bij de implementatie van het spel. In tegenstelling tot de implementatie van het eerste spel (dat een zeer wanordelijke samenstelling had) is het ons gelukt om een duidelijke structuur te bekomen. Hier hebben we ook veel meer kunnen letten op programmatie conventies zoals het vermijden van afhankelijkheden, het systematisch gebruik van macro's enz. Spijtig genoeg is het ons niet gelukt om dit spel correct af te werken en uit te breiden. Dit komt door een tijdstekort zoals beschreven in voorgaande paragraaf.

5.4. *Custom LCD characters*

Om het spel visueel aangenamer te maken, besloten we om aan de 3 mogelijke posities (in GAME_PLAYER) elk een uniek en zelfgemaakt karakter toe te schrijven. Men zou dan kunnen kiezen uit volgende 'avatars'(zie figuur 10).



Figuur 10: Gekozen Custom LCD characters.

Aangezien we een vast aantal custom characters gebruiken in ons project hoeven we geen rekening te houden met de maximale geheugencapaciteit van het CGRAM (waar de custom characters worden opgeslagen) van het LCD-scherm. Deze laatste voorziet namelijk plaats voor 8 custom characters. Eerst zetten we de pinnen van het LCD-scherm juist (waarvoor marco's zijn gedefinieerd in het bestand dwenguinoLCD.h). Vervolgens schrijven we de data van custom character 1, 2 en 3 weg naar respectievelijk het adres 0x40, 0x48 en 0x50 in het CGRAM. Tot slot schrijven we de karaktercode naar het DDRAM.

De implementatie in de code hiervoor is niet gelukt. We hebben dit onderdeel aan de kant laten liggen tot het einde omdat de werking van het spel en zeker de infrarood communicatie zelf ons van groter belang leek. Uiteindelijk hebben dit niet kunnen implementeren door tijdstekort.

6. SAMENWERKING

In de samenwerking ondergingen we een heel proces. We startten met stukken code in mail naar elkaar doorsturen om dan te kopiëren en in te voegen. Al snel toonden de begeleiders ons dat Github een veel efficiëntere manier is. Ook in het gebruik van Github zelf evolueerden we sterk. We leerden werken met branches en oudere versies. Dit groeiproces is nuttig gebleken op het einde van het project en ook in andere vakken hebben we deze kennis reeds kunnen toepassen.

Verder maakten we gebruik van Microsoft Teams om ons elke week voor de on campus sessie op elkaar af te stemmen. We mogen van een vlotte samenwerking spreken.

7. CONCLUSIE

Doorheen het semester hebben we aangeleerd om hardware, in combinatie met software, te hanteren. Op deze wijze hebben we veel geleerd over microcontrollers, maar ook zeker over infrarood materiaal: het gebruik hiervan is vaak een garantie voor een efficiënte en bijna foutloze communicatie. Het eindresultaat van dit proces was de productie van het spel MATCH!. Het feit dat dit spel onafgewerkt is, zorgt voor enige teleurstelling, maar neemt niet weg dat we duidelijke ideeën hebben voor mogelijke afwerkingen van het spel. We zien ruimte voor verbetering: het besturen van het LCD-scherm en het uitbreiden van het spel zijn zaken die wegens tijdsgebrek onafgewerkt blijven.

Daarnaast zou er ook kunnen geworden gewerkt aan andere addities zoals het toevoegen van start- en einde-bits aan de verzonden signalen (voor een vertrouwbaardere communicatie) of aan een versie van het spel dat meerdere beurten heeft.

Dit alles neemt niet weg dat we over het algemeen tevreden zijn over ons eindresultaat.

8. REFERENTIES

[1] Francis Wyffels, Tom Neutens, Victor-Louis de Gusseme, Thomas Lips, *Practicum 1, Project Infra-rood, Logic Analyzer*, ufora, 2022

[2] Francis Wyffels, Tom Neutens, Victor-Louis de Gusseme, Thomas Lips, *Practicum 2*, ufora, 2022

[3] Francis Wyffels, Tom Neutens, Victor-Louis de Gusseme, Thomas Lips, *Project Infra-rood*, ufora, 2022

[4] Francis Wyffels, Tom Neutens, Victor-Louis de Gusseme, Thomas Lips, *Logic Analyzer*, ufora, 2022

[5] Francis Wyffels, Tom Neutens, Victor-Louis de Gusseme, Thomas Lips, *Custom LCD characters*, ufora, 2022

[6] Hitachi, *HD44780U (LCD-II)*, Hitachi

[7] Atmel, *8-bit Atmel microcontroller with 64/128 Kbytes of ISP Flash and USB Controller*, Atmel, 2012.

[8] Vishay, High Power Infrared Emitting Diode, 940 nm, GaAIAs/GaAs, www.vishay.com, 2011

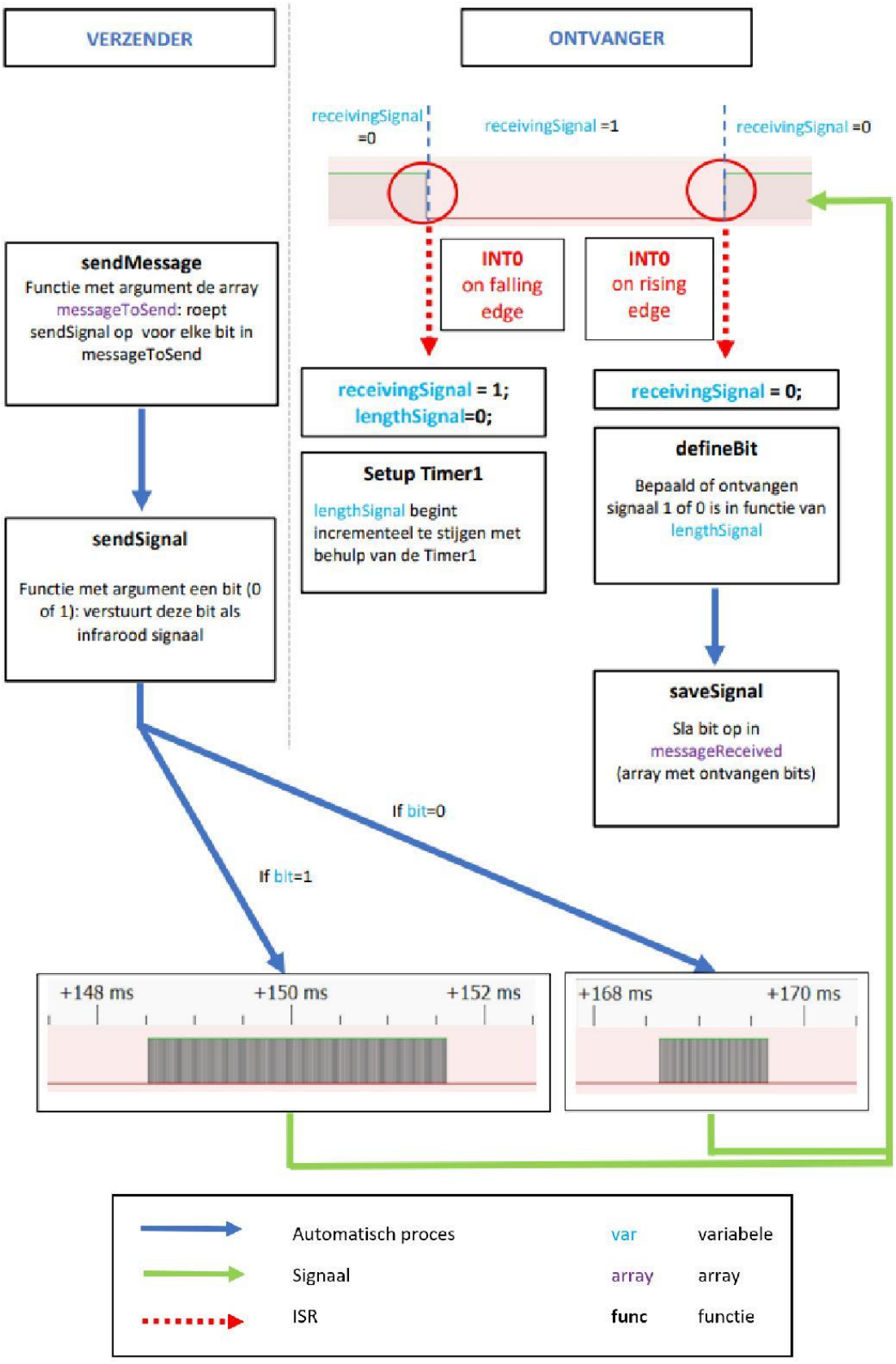
[9] Vishay, IR Receiver Modules For Remote Control Systems, www.vishay.com, 2021

[10] Analogic IC, *PWM: Pulse Width Modulation: What is it and how does it work?*, https://www.analogictips.com/pulse-width-modulation-pwm/, 2017

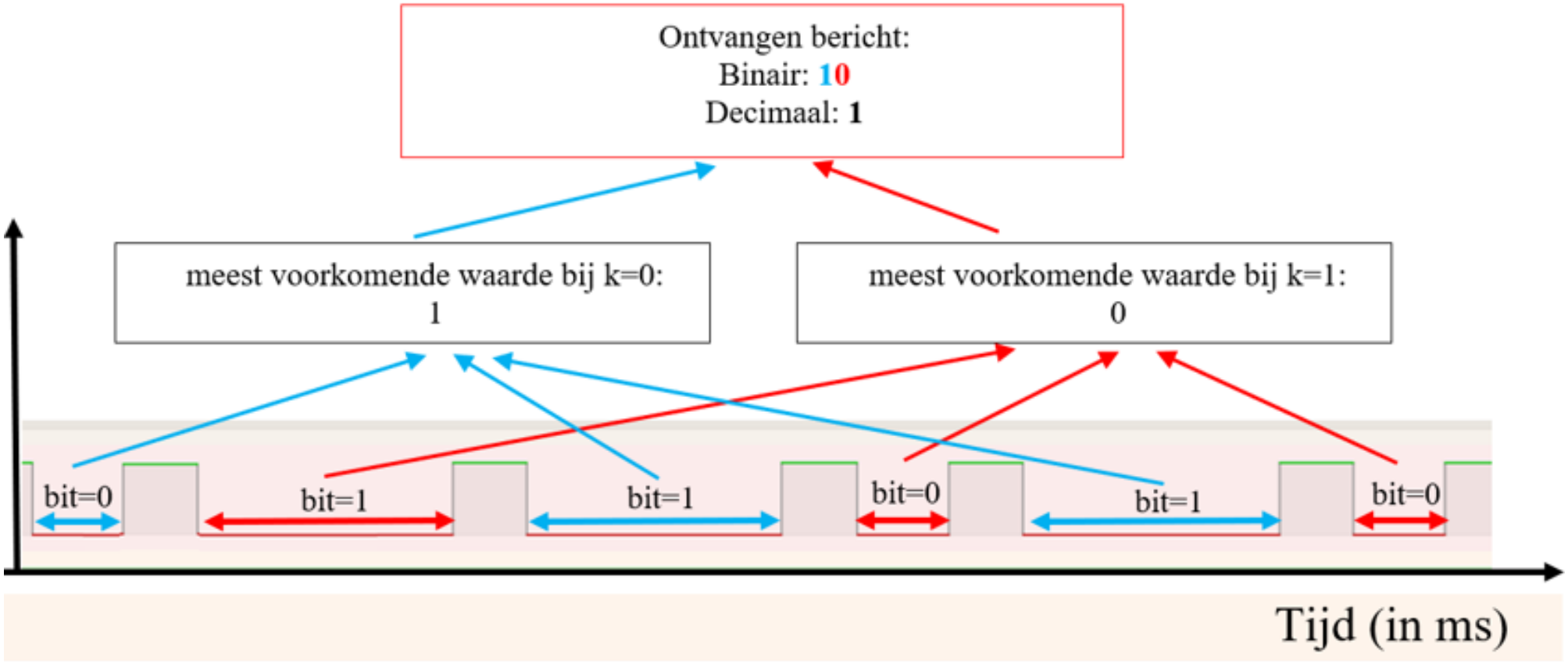
[11] ScienceDirect, *Pulse Width Modulation*, https://www.sciencedirect.com/topics/engineering/pulse-width-modulation, 2014

[12] DigiKey, *Understanding the Basics of Infrared Communications*, https://www.digikey.com/en/maker/blogs/2021/understanding-the-basics-of-infrared-communications, 2021

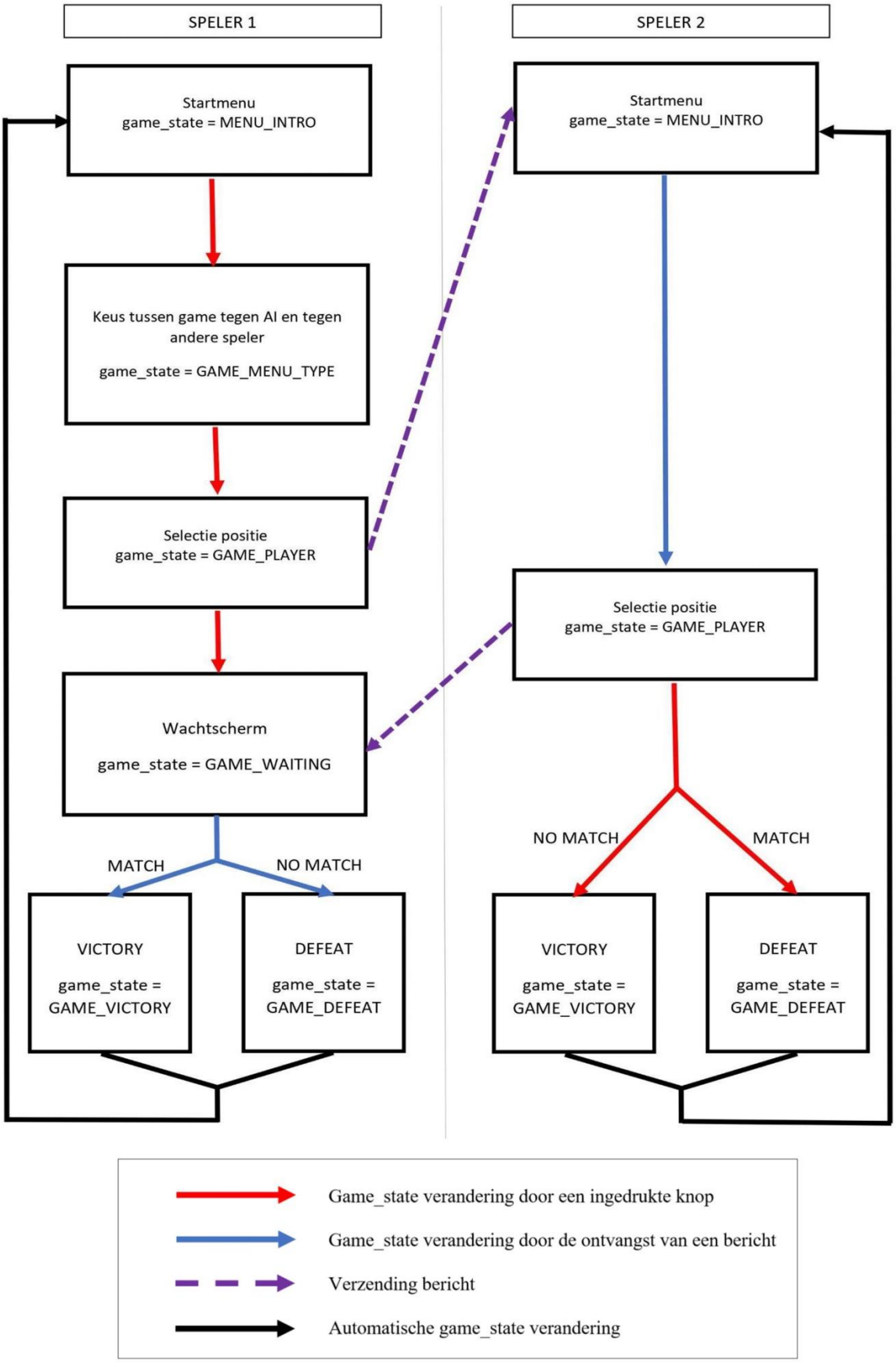
[13] Sparkfun, *IR Communication*, **Error! Hyperlink reference not valid.**



Figuur 3: Schematische representatie van het infrarood protocol



Figuur 8: Schematische representatie van een voorbeeld met repetitiebits.



Figuur 9 : Schematische representatie van de implementatie van de multiplayer-modus in MATCH!