Master of Aerospace Engineering
Internship

Squid Game on GAP – End-to-End Demo Development

S3 Internship report

Author(s): Wendi Ding

Due date of report: 13/10/2022
Actual submission date: 13/10/2022

Starting date of project: 01/April/2022 Duration: 6 Months

Tutors: Antoine Faravelon, Yao Zhang

Table of Contents

Declaration of Authenticity

This assignment is entirely my own work. Quotations from literature are properly indicated with appropriate references in the text. All literature used in this piece of work is indicated in the bibliography placed at the end. I confirm that no sources have been used other than those stated.

I understand that plagiarism (copy without mentioning the reference) is a serious examination offence that may result in disciplinary action being taken.

Date 09/09/2022 Signature 丁文迪

Summary

The goal of this internship is to develop, prototype and test an end-to-end demo application on GAP, namely “1-2-3 GAP” inspired by “1-2-3 sun/grandmother’s footstep” game. The task involves game system design based on GAP processor as well as system implementation and integration with sensors and ext-devices. During the 6-month internship, we first decomposed the game system and investigated existing algorithms to realise the game functionalities. The core algorithms consist of body detection SSD neural network and movement detection through frame difference algorithms. Then we carried out evaluation on the body detection SSD neural network which is under deployment on existing applications on GAP8, and worked on porting the algorithms and peripheral devices (camera and TFT display) to GAP9_EVK, the latest generation of Greenwaves’ GAP processor. As for the extension of the game system, we have added an audio amplifier connected to a speaker to indicate the players about the moving and still phases of the game, and a servo motor fixed with a red laser pointer on its shaft to simulate the gun.

The main difficulties of implementing neural networks on edge devices is the constraint of memory size (L1=128KB, L2=1.5Mb) and computation power. Therefore, we have carefully arranged the memory usage on GAP9_EVK to its limit. To deploy all the selected ext-devices on GAP9_EVK via pre-designed interfaces (MikroE socket, SPI bus, MIPI CSI-2 interface), we have also participated in the development and maintenance of the hardware drivers and APIs (OV9281 camera and ILI9341 display) in Greenwaves’ SDK repository. For the hardware assembly, we have designed and printed a 3D shelf tailored to fix the positions of the board and ext-devices. In the end, the game system is successfully implemented on GAP9_EVK with real-time feedback via sound and display interaction with players at a processing speed of 5fps.

To improve accuracy of the body detection neural network, we have carried out investigation and comparison on existing models from the YOLOX family to find prospective models to replace body detection SSD model. The YOLOX models turn out to have higher accuracy but much slower inference speed (1.5fps). Further training and evaluation of people detection models still needs to be conducted.

The degree of novelty lies in the combination of body detection and frame difference algorithms to detect player’s movement within each bounding box. Besides, as far as we know, the squid game implementation on a SoC processor expanded with a combination of ext-devices to achieve such a real-time interaction with players is also first in the market.

Acknowledgements

I would like to thank all the people who have contributed to the realization of this internship.

My thanks go to my supervisor Antoine Faravelon for the time he has spent on guiding me and teaching me about embedded software development, for sharing his experience and for his continuous help on every aspect of this internship.

I would like to thank Francesco Paci and Ahmad Bijar who have guided me about application development and offered suggestions and devices on neural network investigation. I would like to thank Mickaël Cottin-Bizonne and Paul Luperini who have spent time helping me with Gitlab collaborative version control tool and operation of oscilloscope during this internship. I would like to thank Charlélie Pignol, Nathan Bessieres and Xavier Cauchy for helping me with any hardware issues. I would like to thank Abduragim Shtanchaev and Anastasiya Reshetova from Xperience AI team for providing me with their latest trained YOLOX models for people detection and evaluation dataset.

I would like to thank all the members of the SDK team and AI application team in Greenwaves Technologies for explaining every technical aspect and their latest work to me. I also thank all the people of Greenwaves I have met or contacted online for their warm welcome.

1 Introduction

1.1 Overview of GreenWaves Technologies

GreenWaves Technologies is a fabless semiconductor company founded in 2014 and based in Grenoble, France. The company targets at designing and marketing highly efficient ultra-low-power RISC-V processors for energy constrained products such as hearables, wearables, IoT & medical monitoring products.

The main products of GreenWaves are application processor GAP8 and GAP9. They are designed to interpret and transform rich data sources such as images, sounds and radar signals using AI and signal processing. GAP8 enables embedded machine learning in battery-operated IoT Sensors. It allows image sensors for applications like people counting and attention awareness to run for years on a single AA battery. The second-generation GAP9 processor revolutionizes True Wireless Stereo (TWS) products with ultra-low latency adaptive noise cancellation, neural network-based background noise elimination, multi-channel spatial sound and listening enhancement technologies with market leading energy efficiency.

GreenWaves' system-on-chips enable cooperation companies to develop and bring to market products with new features enabled by state of the art machine learning and digital signal processing techniques. The company also provides leading edge development tools to enable audio and machine learning developers to productively harness the power of GAP processors.

As a growing, talented and highly multicultural team, GreenWaves Technologies insists on non-hierarchical company culture and lives to its core values: ownership, collaboration, agility, dedication to customers and engagement. GreenWaves' team comes from a wide range of backgrounds with more than 10 nationalities. The team cooperates with many companies such as IDUN audio, Orosound and SEGOTIA and provides technical support with GAP processors, bringing successful new-to-world products to the market.

1.2 Product Overview

Table 1 shows the product overview of ultra-low-power GAP processors.

Table 1: product overview of ultra-low-power GAP processors

	GAP8	**GAP9**
Processing power	22.65 GOPS	150.8 GOPS
Power efficiency	4.24 mW/GOP	0.33 mW/GOP
Memory		
L1	80 kB 128 kB	
RAM	512 kB 1.5 MB	
Non Volatile	None	2 MB
External	QSPI/ HyperBus	2xQSPI/OCTO-SPI/HyperBus/SDIO
MAX Frequency		
FC*	250	400
Cluster**	175	400

Fixed Point	8, 16, 32-bit	8, 16, 32, 64-bit***
Floating Point	None	16/16alt/32
Sound Interface	2 Rx-Only I2S interfaces	3 master/slave SAI full duplex, I2S and TDM 4/8/16 channel capable
Camera Interface	8-bit CPI (Camera Parallel Interface)	8-bit CPI, 2-lane CSI-2
Package type	aQFN 88 7x7mm	WL-CSP 3.7mmx3.7mm – BGA 5.5mm x 5.5mm

*FC (Fabric Controller) is the main system controller and resembles a standard MCU; it can delegate compute-intensive tasks to the Cluster.

**GAP includes a multicore compute cluster with a shared memory architecture and hardware thread synchronization. The cluster enables highly efficient, parallel implementation of algorithms giving almost optimal linear speedup.

*** 64 bit support included in selected instructions.

The GAP9 platform is exceptionally power efficient for voice, music and image processing. It gives a headroom in both energy and processing power that can be used to develop innovative new features with no compromise in area, cost or energy. GAP9 also allows for multi-sensor analysis (vision and sound) and is a perfect solution for battery-powered smart security systems and smart buildings. The GAP9 processor supports always-on, battery-operated inference on images, sounds and more, including:

- Occupancy management
- Surveillance systems
- Face detection / identification
- Speaker detection / identification
- Voice driven user interfaces

1.3 GAP9 Processor Architecture and Features

GAP9 is a unique combination of a powerful low power microcontroller, a programmable compute cluster with a hardware neural network accelerator and sample by sample audio filtering unit. This combination of homogeneous processing units with integrated hardware acceleration blocks achieves a perfect balance between ultra low power consumption and latency, flexibility and ease of use.

As shown in Fig. 1, all the 10 cores in GAP9 are based on the RISC-V Instruction Set Architecture (ISA) extended with custom instructions automatically used by the GAP toolchain. The compute cluster is adapted to handling combinations of neural network and digital signal processing tasks delivering programmable compute power at extreme energy efficiency. The architecture employs adjustable dynamic frequency and voltage domains, and automatic clock gating to tune the available compute resources and energy consumed to the exact requirements at any particular moment. GAP9's revolutionary Smart Filtering Unit (SFU) is perfectly adapted to ultra low latency (1us) PDM to PDM filtering tasks. It has a rich set of interfaces including 3 Serial Audio Interfaces (SAI) capable of handling up to 48 incoming or outgoing audio signals. GAP9's hierarchical and demand-driven architecture is perfectly suited to design the next generation of hearable products and applications for battery-powered smart sensors.

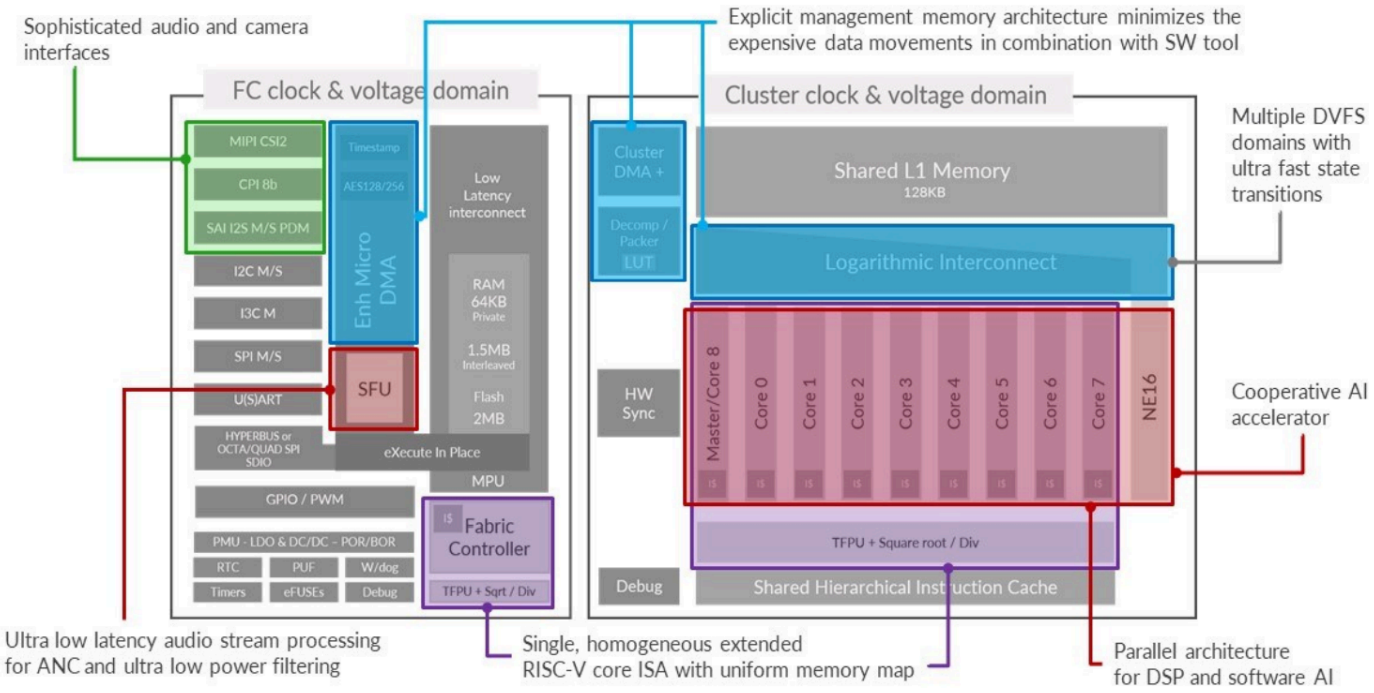


Figure 1: GAP9 processor architecture

Hardware features of GAP9 processor are listed below:

- 9 core RISC-V compute cluster with AI accelerator and 1 core RISC-V controller
- Smart Filtering Unit (SFU) for sample by sample 1.3µs audio filtering
- Transprecision floating-point support (IEEE 32-bit and 16-bit and bfloat16)
- On-the-fly hardware AES128/256 encryption / decryption and PUF
- 1.6MB retentive L2 RAM
- 2MB non volatile memory
- Optional external high-speed low power SDRAM and Flash with mappable virtual memory support
- Multi channel PCM/PDM SAI interfaces
- CSI-2 camera interface
- WL-CSP 3.7mm x 3.7mm - BGA 5.5mm x 5.5mm package options

The following indices demonstrate the performance of GAP9 processor:

- System performance of 330µW/GOP
- Up to 370 MHz internal clock
- Up to 15.6 GOPs DSP and 32.2 GMACs machine learning
- Deep Sleep: ~45uW, wake up time : ~2.5ms
- Integrated LDO/DC-DC
- 1.8V-I/O voltage
- 1.8V-5.5 V regulator supply voltage
- 0.4ms cold boot time
- 2µs to power and start cluster

1.4 Development Toolset

Greenwaves' State of the art toolchain with a wide range of supported models and frameworks makes development significantly easier, including:

- RISC-V C / C++ toolchain with full support for ISA extensions based on GNU toolset (GCC & GDB)
- FreeRTOS™ support
- GAP AutoTiler code generator for explicit memory movement
- GAPflow tools providing end-to-end code generation from Neural Network (NN) frameworks such as TensorFlow and PyTorch
- Cross OS PMSIS cluster & device API
- GVSOC System on Chip (SoC) simulator and visual code profiler
- Debug support including on-chip debug

GVSOC is a lightweight and flexible instruction set simulator which can simulate Greenwaves' GAP9 IoT application processor. It allows execution of programs with real device drivers on a virtual platform. In this project, we mainly use GVSOC simulator to simulate the on-board application, and we deploy FreeRTOS and PMSIS APIs to arrange synchronous/asynchronous tasks among the 9 cluster cores and Fabric Controller (FC). The PMSIS API is a set of low-level drivers that any operating system can implement to provide a common layer to upper layers, allowing the development of applications portable across a wide range of operating systems. In addition, the GAPFlow which consists of NNTool and Autotiler is also used in this internship to read tflite/onnx graphs, quantize the NN models and generate executable application codes for testing on GAP processors. Fig. 2 shows the procedures of the GAPFlow toolchain.

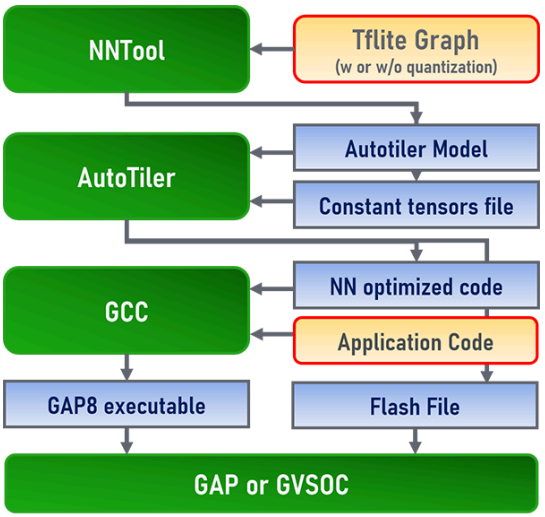


Figure 2: GAPFlow toolchain for neural network implementation on board

2 Context and key issues

This 6-month internship aims at developing, prototyping and testing end-to-end demo application on GAP, “1-2-3 GAP”, inspired by “1-2-3 sun/grandmother’s footstep” game. The task involves:

- System design: design the game system from A to Z based on GAP processor.
- According to the system, reuse and extend existing algorithms and codes.
- System implementation (C programming), with sensors and ext-devices.
- Potentially propose extension, on either software or hardware perspect for the demo.
- Shoot marketing video for GAP business promotion.

2.1 Project breakdown

The system is based on GAP9_EVK, an Evaluation and Development Kit for GreenWaves Technologies's GAP9 chip. Fig. 3 shows the requirement diagram of "1-2-3 GAP" game system. It mainly consists of three tasks:

- Detection of players' position and movement
- Announcement of moving and still phases
- Display of real-time video record on screen.

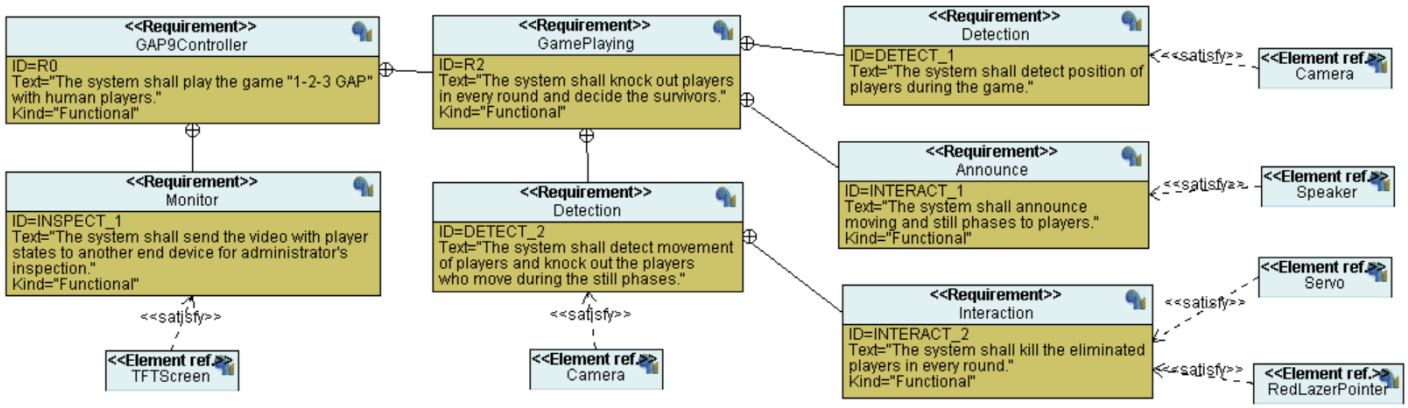


Figure 3: Requirement Diagram of "1-2-3 GAP" game system

Because of hardware/software limitations during is internship, the game system is built based on the following assumptions:

- The players shall not hide behind, come across or come too close to each other.
- The players who are ruled out by the game system in each round should be "killed" and quit the game immediately.
- The camera is in a fixed position and the environment change of luminosity is minor.
- The power supply is always on and all peripherals are in good connections.

2.2 System Design

The requirements are realised with extra hardware devices including TFT screen, camera, speaker, servo and red laser pointer. Fig. 4 shows the Use Case Diagram of the game system. The game system is composed of 4 main functions: AnnounceGamePhases, RuleOutPlayers, DetectPlayerMovement and SendImagesToAdministrator, realised by extra devices i.e. speaker, servo, red laser pointer, camera and TFT screen respectively.

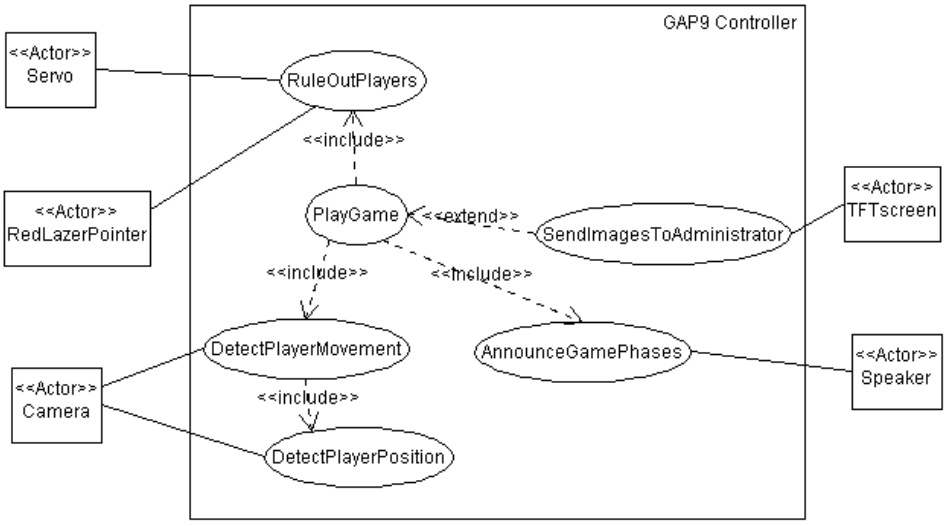


Figure 4: Use Case Diagram of "1-2-3 GAP" game system

The Block Diagram of the game system is shown in Fig. 5. The interfaces of the game system include sound record play, image capture, red laser pointer on/off control, pwm signal on/off control and display control. Internal methods of GAP9 Controller include resize captured image, send Convolution Neural Network (CNN) inference task to cluster, send Single Shoot Detection (SSD) processing task to cluster, compute frame difference, draw bounding boxes on image, etc.

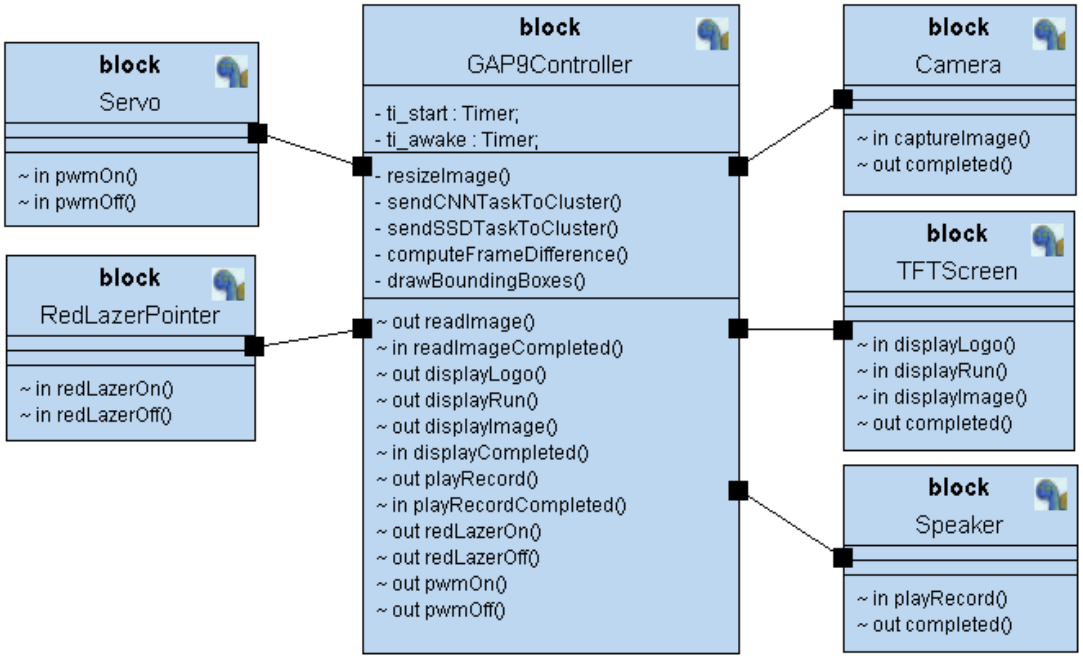


Figure 5: Block Diagram of "1-2-3 GAP" game system

Fig. 6-7 shows the Finite state machine of GAP9 main controller (without servo and red laser pointer control) and the other components. The game loop is divided by alternative moving phase and still phase. Starting with the moving phase, GAP9 controller plays the game sound record. When the sound ends, the game will enter the still phase. The players are supposed to keep their post and position still during a still period of 5-10 seconds. During this period, we have an inner loop where the camera keeps on capturing images, and the GAP9 controller preprocesses the

images, sends CNN and SSD tasks to the cluster cores on board, which are in charge of inferencing body detection neural network.

The output of the body detection SSD neural network is the position of bounding boxes (x,y,w,h), with its score, class and status. If there exists an image from the previous loop (imageId > 1) during the same still phase, then a frame differencing computation is performed on the previous image before inferencing NN models on the current image. The frame differencing computation will give a judgment of whether the player inside each bounding box has moved or not. If the player has moved, then the status of the bounding box is set to “dead”, and in the next step a cross will be drawn across this bounding box on the previous image. On the other hand, if no movement is detected, the player survives this round of detection and the status of the bounding box is kept “alive”. In this situation, the bounding box will be drawn on the previous image without the cross. After drawing the bounding box, the previous image will be sent to the TFT screen to be displayed.

A time counter (ti_awake) is always initialized when the game system enters the still phase. When time is up for the still phase, the controller will enter the next game loop starting with the moving phase again. The total time limit of the game is determined by a user-predefined GAME_TIME. When GAME_TIME is up, the game is over and all the rest players still within the scope of the camera are ruled out.

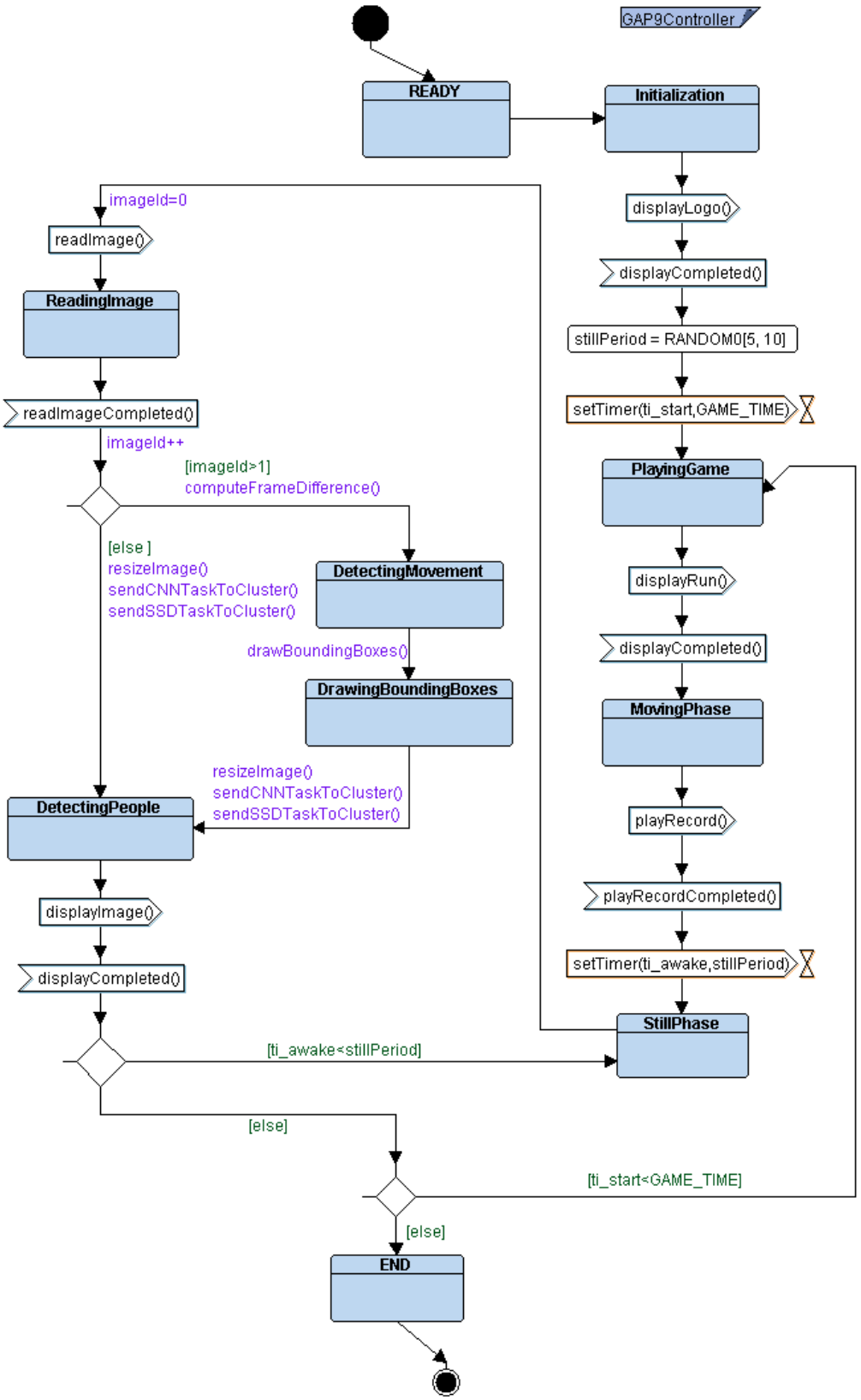


Figure 6: Finite state machine of GAP9 main controller

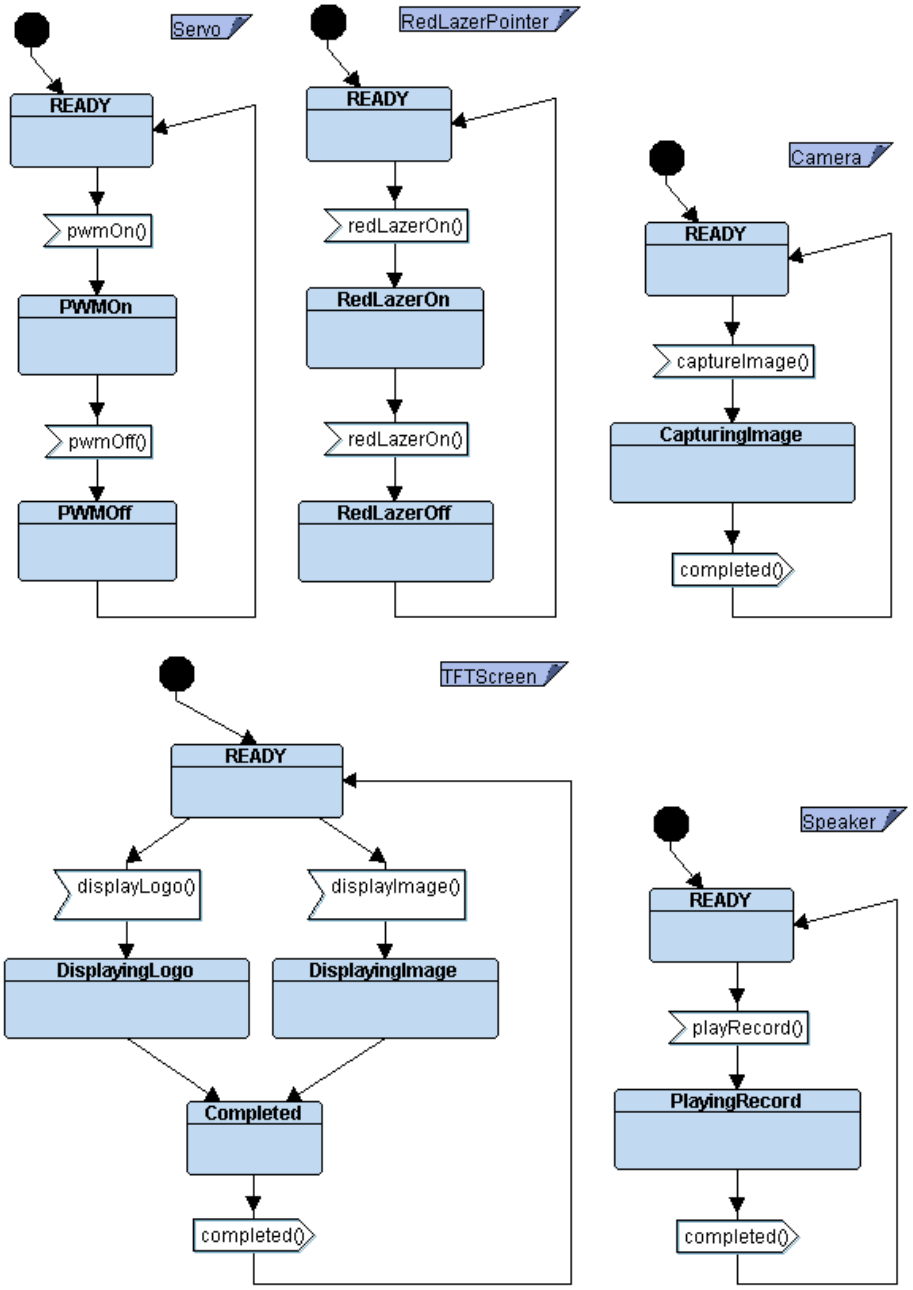


Figure 7: Finite State Machine of "1-2-3 GAP" game system components

The Sequence Diagram of the "1-2-3 GAP" game system in Fig. 8 shows the real-time interaction between GAP9 controller and extra devices. For simplification we neglect the initialization interactions which will be demonstrated in detail in **Section 3.5 - Software Integration**. The hollow arrow indicates asynchronous message, which controls the servo and red laser pointer to run in parallel with the other algorithms.

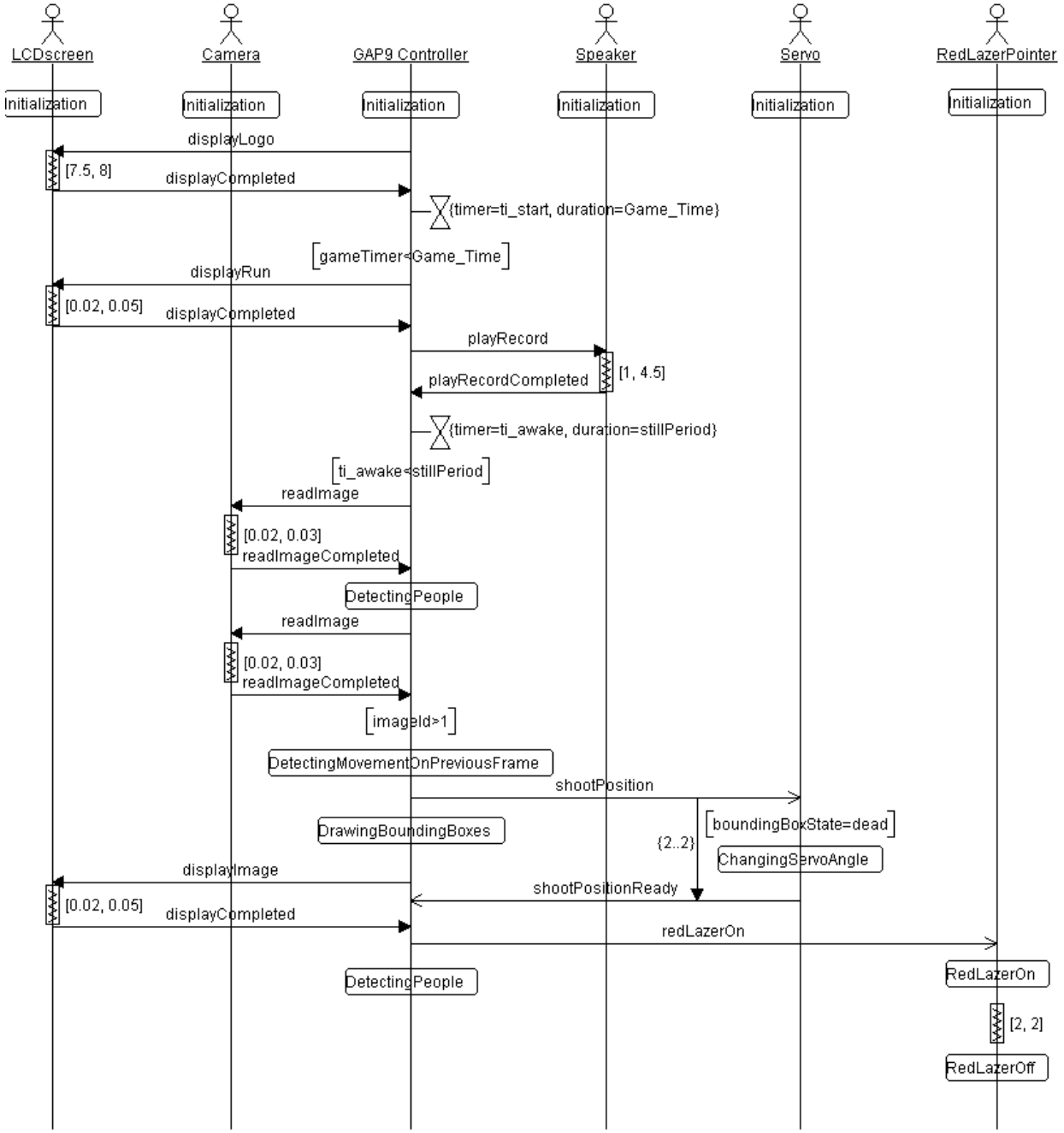


Figure 8: Sequence Diagram of "1-2-3 GAP" game system

3 Investigation Methods

3.1 Survey on Neural Network

For the task of people detection, we have made a survey on the neural network approaches trained for the purpose of object detection in computer vision. People detection task aims at locating the presence of people with bounding boxes with confidence scores in an image. The performance of an object detection model is evaluated using the precision and recall across the best-matching bounding boxes for the known objects in the image. Precision indicates the ability of a model to identify only relevant objects (fewer false detections), while recall indicates the ability of a model to find all ground-truth bounding boxes. Mathematical definitions of precision and recall are demonstrated in equations below [1] :

$$Precision = \frac{TP}{TP+FP} = \frac{TP}{all\ detections} \qquad TP = True\ positive \quad (1)$$

$$Recall = \frac{TP}{TP+FN} = \frac{TP}{all\ ground\ truths} \qquad FN = False\ negative \quad (2)$$

Where true positive (TP) means the number of correct detections (Intersection over Union (IoU) > a certain threshold) of an object of a certain class, while false negative (FN) means the number of non-detection of an object of a certain class (IoU ≤ a certain threshold) in an area that contains no object of such class as a matter of fact. IoU measures how much the predicted bounding box overlaps with the ground truth divided by the area of union between them.

The general definition for the Average Precision (AP) is the area under the precision-recall curve (see Equation (3)). A high AP tends to indicate a better trade-off between precision and recall.

$$AP = \int_0^1 p(r)dr \quad (3)$$

Where r is recall and p(r) means precision at the recall value of r. The mAP (Mean Average Precision) is the average of AP over all classes of detection.

In practical cases, the precision-recall curve is often a zigzag-like curve, posing challenges to an accurate measurement of AP. In this internship, we use the approach of all-point interpolation to approximate the precision-recall curve and to facilitate the calculation, as shown in Equation (4).

$$AP = \sum(r_{n+1} - r_n)p_{interp}(r_{n+1}) \quad (4)$$

$$where\ p_{interp}(r_{n+1}) = max\ p(\hat{r}),\ \hat{r} \geq r_{n+1}$$

Deep-Learning-based detection methods for object detection are classified into two-stage detectors and single-stage detectors. State-of-the-art models are listed below:

- Two-stage detectors: R-CNN (Region-Based Convolutional Neural Network), Fast R-CNN, and Faster-RCNN
- Single-stage detectors: YOLOv3,v5,v7 (You only Look Once), YOLOX, SSD (Single Shot Detector)

The R-CNN models are generally more accurate, however, these models are proved to be too slow for real-time inference (see Table 2).

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Table 2: Results on Pascal VOC2007 test (4952 images) on a Nvidia Titan X [2].

The YOLO family and SSD models are designed for speed and are promising to achieve real-time people detection in this internship project. The key difference between the two architectures is that

the YOLO architecture utilizes 2 fully connected layers, whereas the SSD network uses convolutional layers of different sizes. The ready-to-use people detection neural network trained and quantized by Greenwaves' AI team is based on the concept of SSD network. It is extremely light-weighted and is composed of only 32 layers (28 convolutional layers + 4 max pooling layers). The model has already been quantized with scaled scheme and transformed into application code to be deployed on GAP8 platform.

Due to external device and memory constraints (will be explained in **Section 3.5**), we have to reduce the input image size as much as possible while maintaining acceptable detection accuracy and real-time inference speed on edge devices, i.e. GAP9_EVK. In order to fit in a 1.5MB L2 memory, we require the maximum input image size to be smaller than 640x400x1, say 0.256MB. Since the mainstream of CNNs is trained on 3-channel RGB images with a relatively large input size to achieve high mAP performance but not designed for real-time inference on edge devices, we have to carefully select suitable people detection models according to their input image size, Gflops, number of parameters and model size.

Considering that the training of CNNs is beyond the scope of this internship project, we have consulted Xperience AI, a cooperation company of Greenwaves that trains accurate and fast neural networks ready to be deployed on all platforms, to acquire their state-of-the-art people detection models customized for real-time inference on edge devices.

The models are trained on the backbone of yolox_tiny and yolox_nano, fast and light-weighted models from the yolo family. The training and evaluation dataset is COCO 2017 database [3] filtered on person. The performance (AP, Gflops) evaluation made by Xperience AI is listed in Table 3. For comparison, we have also made a general evaluation of the current SSD model under deployment in Greenwaves and added the evaluation results in Table 3. This SSD model was trained on MPII Human Pose Dataset [4] in 2019 and has been deployed on GAPUINO GAP8 for the body detection application. The reference of body detection SSD model is carried out on GVSOC which simulates running the model on GAP9 platform, and the detections are saved in the script file output.txt for later evaluation with eval.py. We set $IoU\ threshold = 0.3$ as the criterion to categorize the detected bounding boxes as "TP" or "FP", depending on their IoU values with ground truth boxes. The test has been carried out on 5000 images of COCO 2017 evaluation dataset filtered on people, and the precision-recall curve is drawn in Fig. 9 with the original curve in blue and all-point interpolation curve in orange. We have calculated the area under the interpolated curve to be 0.198 at $IoU\ threshold = 0.3$.

Table 3: Performance evaluation of SSD and YOLOX models

model name	input image type	input image size	AP@0.5	AP@0.5-0.95	Gflops / GMac*	Parameter Cont.	Size MB
yolox-nano	RGB	320x240	0.887	0.601	1.85 / 0.93	0.90M	7.3 MB
yolox-nano	RAW Bayer	320x240	0.88	0.606	1.85 / 0.94	0.90M	7.3 MB
yolox-nano	RAW Bayer	640x480	0.858	0.557	1.90 / 0.95	0.90M	7.6 MB
yolox-tiny	RAW Bayer	640x480	0.742	0.453	11.09 / 5.55	5.06M	5.03M
body_detection SSD	Grayscale	160x120	0.151	0.053	-	0.04M	-

*FLOPs (floating point operations) is used to describe how many operations are required to run a single instance of a given model. Mac is the number of multiply-accumulate operations for a single convolution layer.

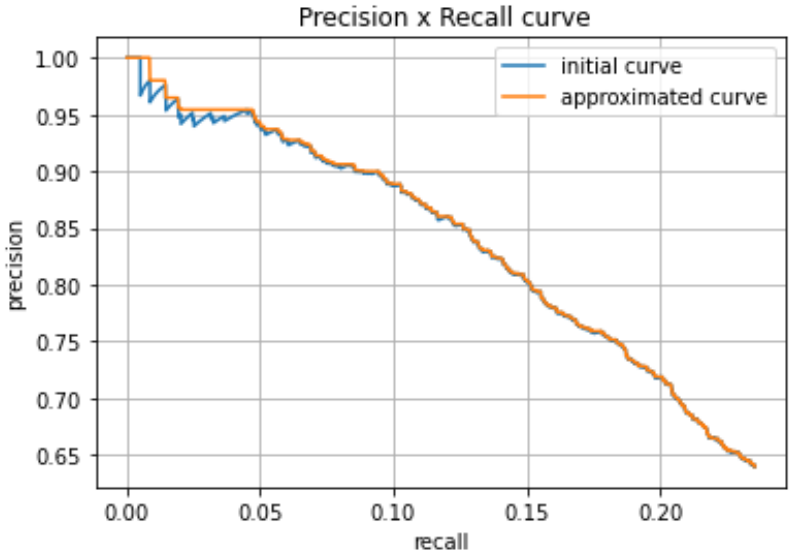


Figure 9: Precision-recall curve with IoU threshold=0.3. The original curve is in blue and the all-point interpolation curve is in orange.

From the model evaluation, we can find that the body detection SSD model has a rather low AP compared with yolox models. The reason lies in the smaller input image size (only 160x120) and much fewer layers and parameters in the neural network, as well as the fact that the body detection SSD model is trained on another dataset than filtered COCO 2017, and that the model has already been quantized. However, we are still going to use this body detection SSD model in the first place because it is a ready-to-use model tailored for GAP platforms and because it has already been proved to be feasible to deploy on GAP8 for the body detection application.

3.2 Investigation on Movement Detection Methods

Movement detection methods are employed to detect players' movement during still phases of the game. The idea is to use frame differencing methods within each bounding box surrounding the players to detect the players' movement. Traditionally, movement detection methods are classified into three categories:

- **Frame Difference Method**

  The frame difference algorithm compares every pixel within 2 consecutive frames via the difference of pixels. A threshold is set to distinguish between "motion" and "stillness" of pixels, which are then converted into a binary map. This basic algorithm is called two-frame difference, and an improvement of the two-frame difference method is the three-frame difference method. It puts three adjacent frames as a group, subtracts both two adjacent frames and does an logical AND operation between the two differential results.

- **Background Subtraction Method**

  In the Background subtraction method, the current frame is subtracted with a reference background frame calculated from the initial frames in order to find the moving foreground object. This method is suitable for processing images from static cameras, but is susceptible to the change of luminosity or exposure time.

- **Optical Flow Method**

Optical-flow methods are based on computing estimates of the distribution of apparent velocities of movement of brightness patterns in an image.

Among the above-mentioned algorithms, the frame differencing method is suitable for a variety of dynamic environments, but generally it is difficult to get the complete outline of the moving object. Three-frame difference method improves detection accuracy but also introduces time and space complexities compared with the two-frame difference method. In contrast, large quantity of calculation, sensitivity to noise, poor anti-noise performance makes optical-flow methods not suitable for real-time demanding occasions. The background subtraction method has a simple algorithm, but is very sensitive to the changes in the external environment and has poor anti-interference ability [5].

Through building the movement detection algorithm on top of the output bounding boxes of people detection CNNs and taking only the pixels within the bounding boxes into account, we can exempt from the task of localization of the moving objects and thus reduce the computation complexity as well as the cruciality of movement detection algorithms' precision. Taking into consideration the L2 memory size of 1.5MB on GAP9_EVK and real-time inference requirement, we first choose to take the algorithm of two-frame differencing because of its simplicity, provided that the camera is fixed.

To carry out the test on frame differencing algorithm, a batch of RGB images captured from OV5647 Rpi camera v2 is resized, converted to grayscale and fed into the SSD neural network to obtain the bounding boxes for each image. Then we apply the two-frame differencing method to each two consecutive images to obtain the absolute difference between them. The thresholds of 0, 10, 20, 30, 50, 70 are applied to the absolute difference respectively to get the binary image. Finally, the binary pixels are compared with bounding box positions, thus indicating whether the players inside the bounding boxes have moved or not. Fig. 11-13 shows the test result on three typical binary images: DIFF_4, DIFF_13 and DIFF_51.

Figure 10: Absolute frame difference images DIFF_4 (g), DIFF_13 (h) and DIFF_51 (i) and their corresponding consecutive two frames (a and d, b and e, c and f).

In Fig. 10, we note that in the first two frames (see Fig. 10(a, d)), player 1 has a slight movement in the upper body; in the second two frames (see Fig. 10(b, e)), player 2 has a remarkable movement while player 1 has a slight movement in the upper body; in the last two frames (see Fig. 10(c, f)), player 3 appears on the image, player 2 and player 3 have remarkable movement while player 1 has noticeable movement as well.

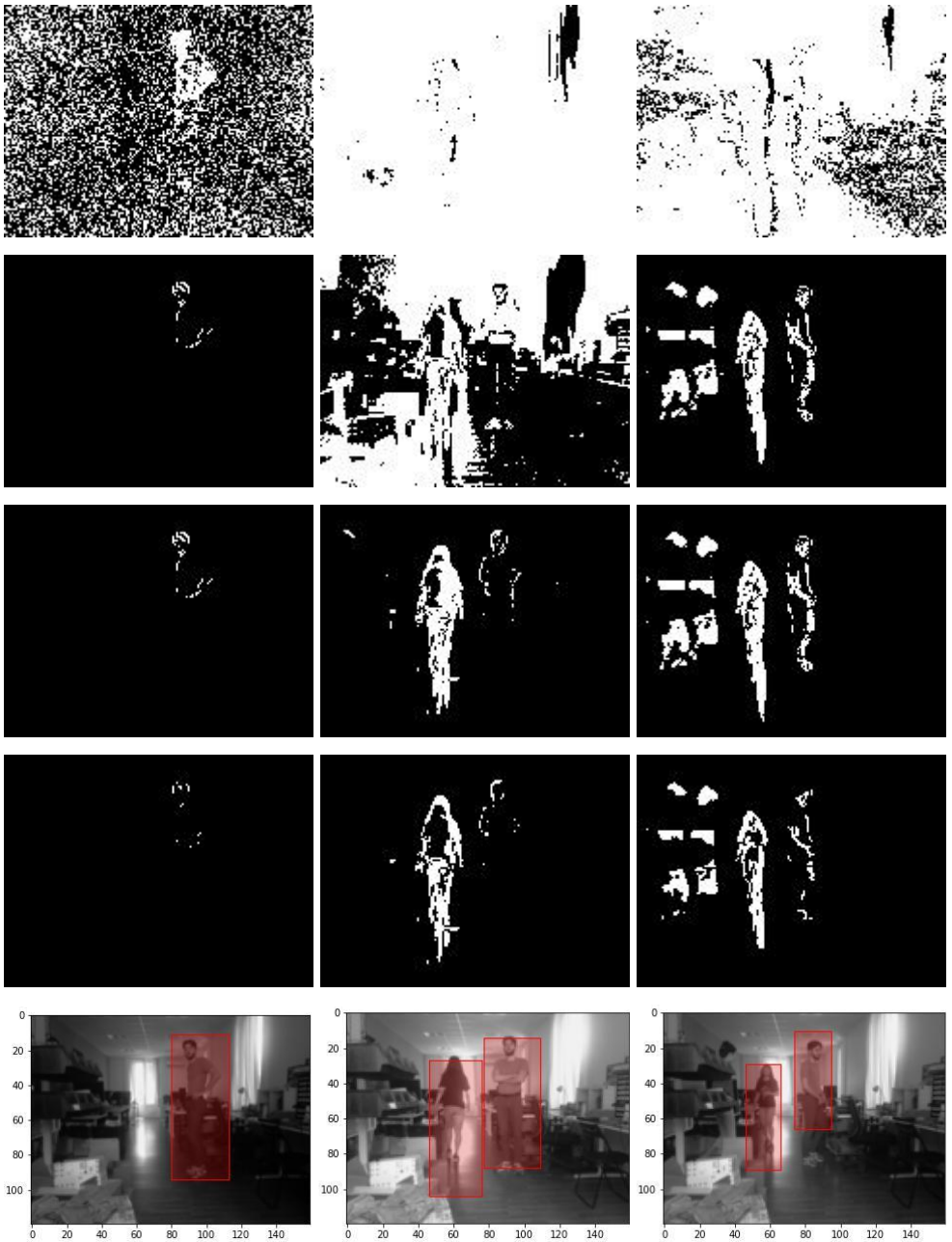


Figure 11: Binary images THRESH_4 (a,d,g,j), THRESH_13 (b,e,h,k) and THRESH_51 (c,f,i,l) calculated from absolute frame difference with thresholds of 0 (a-c), 10 (d-i) and 20 (j-l), respectively. At threshold=10, an offset equal to the average frame difference is applied to obtain the binary images (g-i). The results of movement detection are drawn in images (m-o).

In Fig. 11, we can see that the background noise is dominant when the binary threshold is set to 0 (see Fig. 11(a-c)), the noise is reduced and avoided when we increase the threshold to 10 and 20 (see Fig. 11(d-f, j-l)). However, when the luminosity changes between the frames, greater background noise will be introduced in the frame difference (see Fig. 11(e)). To deal with this issue,

we reduce the frame difference by the average frame difference of the whole image to offset the global change in luminosity. Fig. 11(h) demonstrates the effect of background noise reduction compared to Fig. 11(e), where the movement is notably captured with the white pixels.

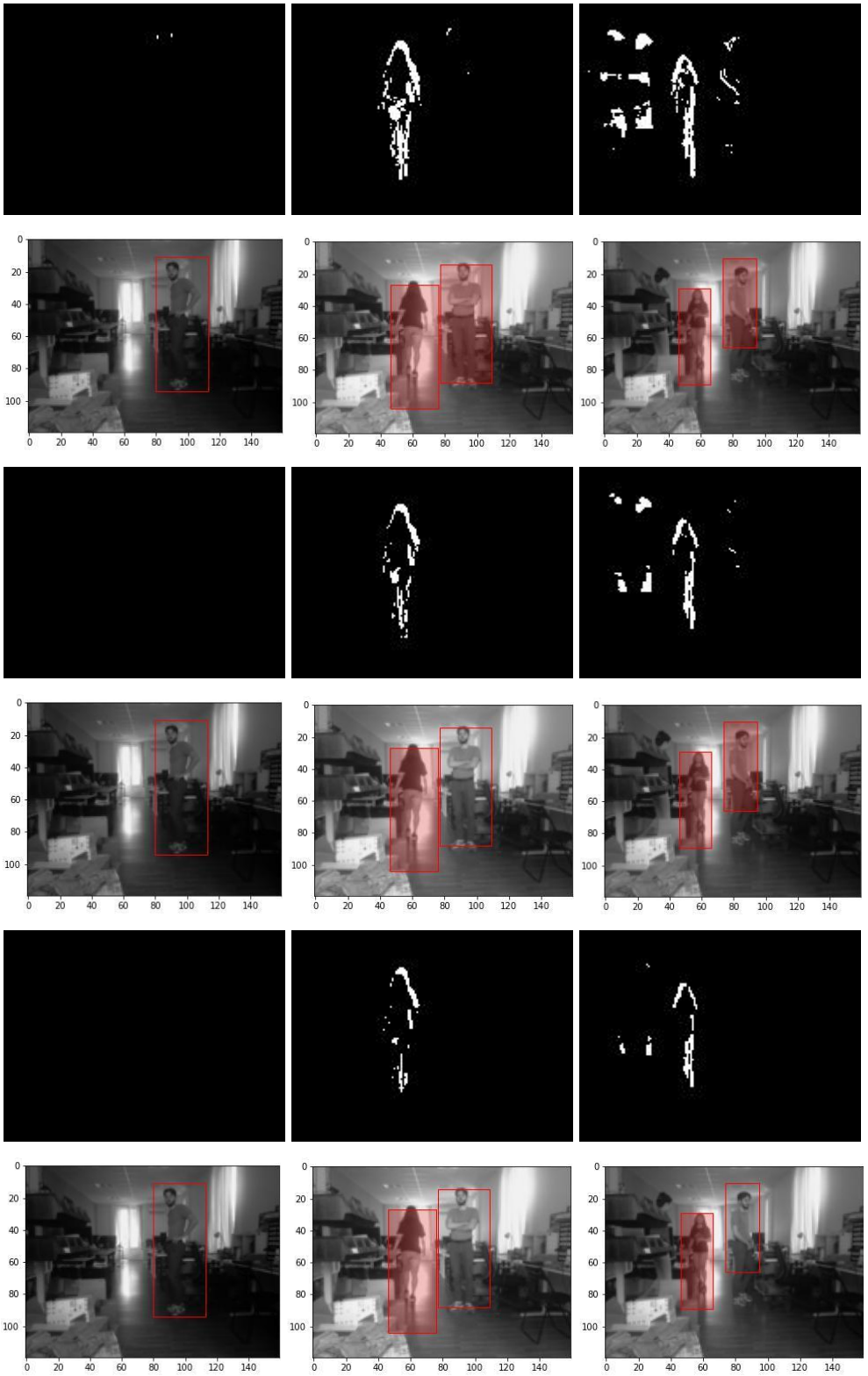


Figure 12: Binary images THRESH_4 (a,g,m), THRESH_13 (b,h,n) and THRESH_51 (c,i,o) calculated from absolute frame difference with thresholds of 30 (a-c), 50 (g-i) and 70 (m-o), respectively. The results of movement detection are drawn in images (d-f, j-l, p-r).

Fig. 12 shows the effect of increasing the frame difference threshold up to 70. Slight movement of player 1 is omitted when frame difference threshold is equal to 30 (see Fig. 12(a,d)) and 50 (see Fig. 12(h,k)), while the remarkable movement of player 2 is always detected. However, the noticeable movement of player 1 in the third frame difference image is omitted when threshold=70 (see Fig. 12(o,r)). In order to omit slight movement and to capture noticeable movement of the players at the same time, we set as a compromise solution that frame difference threshold equals to 50 in the movement detection algorithm of the game system. In addition, an offset of the average frame difference value is applied to balance the change of luminosity.

To sum up, the limitations of the frame differencing algorithm based on SSD model output bounding boxes lie in the following three points:

- The noise introduced by the change of luminosity can influence the movement detection.
- The camera must be fixed to ensure a valid movement detection.
- The movement of one player can influence the movement detection judgment of the other player if the two players approach too close or pass by each other.
- If one player changes the position too fast compared with game algorithm computation time, the game system can lose track of the player's movement.
- If one player hides behind an object (see player 3 in Fig. 11-12) or stands too far from the camera, or the color of the player's clothing merges into the background, the player can be undetected.

3.3 Hardware Selection

3.3.1 GAP9 Evaluation and Development Kit (GAP9_EVK)

GAP9_EVK is an Evaluation and Development Kit for GreenWaves's GAP9 chip. It is an ultra-low power application processor for edge AI and Digital Signal Processing (DSP). GAP9_EVK is intended for generic evaluation of GAP9 capabilities. Application-focused developments will typically require coupling GAP9_EVK with extra hardware, for instance some sensors or an audio board. Multiple expansion connectors are provisioned on GAP9_EVK (see Fig. 13).

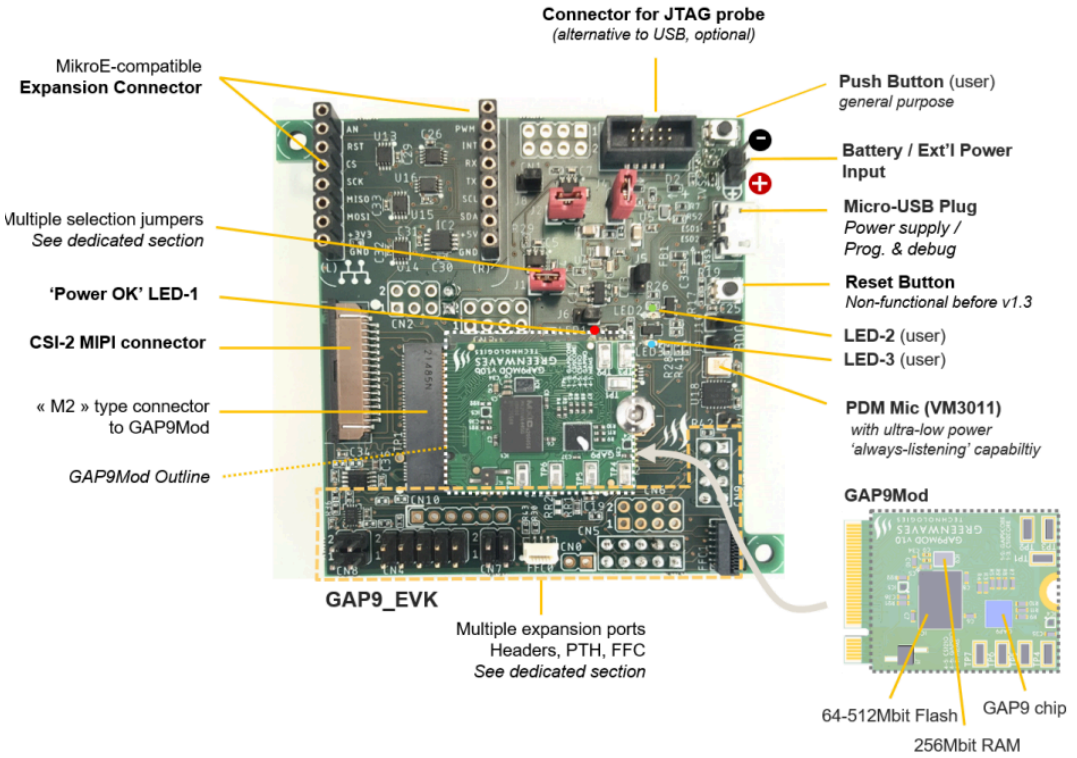


Figure 13: GAP9_EVK board overview.

The central module of GAP9 is GAP9Mod. It consists of the GAP9 chip, a 64Mb/512Mb Octal SPI Flash and a 256Mb Octal SPI RAM. The MikroBUS socket is intended to plug the MikroE Click board. It consists of 3 groups of communication pins (SPI, UART, I2C) , 6 additional pins (PWM, Interrupt, Analog Input, Reset and Chip Select) and power supplies. The pins of the MikroBUS connected to GAP9 I/Os are shown in the mapping in **Appendix 1**.

In this project, GAP9_EVK serves as the main controller of the game. We use the MikroE-compatible interface to control the display via SPI bus, and use CSI-2 MIPI interface to control the camera. The servo motor is controlled by a PWM signal generated on MikroE bus, with the PWM pin configured to the GPIO output function. The red laser pin is controlled by the RST pin also configured to GPIO output. We choose to use MikroE interface because it is built with a level shifting circuit such that it can output 3.3-5V signals, which is compatible with the input voltage of Adafruit TFT display, servo motor and red laser pointer. In the meanwhile, the other pins on GAP9_EVK only outputs 1.8V signals.

The CSI-2 MIPI (C-PHY) interface of GAP9 (only 1-lane mode supported by GAP9 in WLCSP package) is routed to a 15 pin FFC (Flat Flexible Cable) connector compatible with Raspberry Pi camera modules, making it possible to work with readily available Rpi cameras.

3.3.2 Camera

Since the CSI-2 MIPI (C-PHY) interface of GAP9 in WLCSP package only supports 1-lane mode, at present GAP library only supports device API for cameras OV9281and OV5647 in 1-lane mode:

- The ‘official’ RaspberryPi camera v1: 5 Mpix, color, based on Omnivision’s OV5647
- Arducam’s OV9281 module : global shutter, monochrome, 1 Mpix

Product specifications of the cameras are listed in Table 4 below.

Table 4: Product specifications of cameras OV9281 and OV5647

Sensor	Monochrome global shutter OV9281	Color CMOS 5-megapixel image sensor OV5647
Pixel Size	3 µm x 3 µm	1.4 µm x 1.4 µm
Active array size	1296 x 816	2592 x 1944
Output interface	2-lane MIPI serial output and DVP parallel output	2-lane MIPI serial output
Output formats	8/10-bit raw BW data	8/10-bit raw RGB data
Maximum image transfer rate	1280 x 800@120 fps	- 1080p: 30 fps - 960p: 45 fps - 720p: 60 fps - VGA (640x480): 90 fps - QVGA (320x240): 120 fps
Input clock frequency	6~27 MHz	6 ~ 27 MHz
Support for image sizes	- 1280 x 800 - 1280 x 720 - 640 x 480	- 1080p (1920x1080) - 960p (1280x960) - 720p (1280x720)

	- 640 x 400	- VGA (640x480) - QVGA (320x240)

In this project, we choose monochrome global shutter OV9281 for on-board implementation. OmniVision's OV9281 is a high-speed global shutter image sensor that brings 1-megapixel resolution. It captures 1280x800 resolution images at 120 frames per second (fps) and VGA resolution at 180fps with 2-lane MIPI and DVP output. The OV9281's high frame rates make it an ideal solution for low-latency machine vision applications. Besides, the camera's wide FoV satisfies the requirement of monitoring the game. Moreover, OV9281' output of grayscale images corresponds to the input format of our SSD neural network so that we can exempt from complex pre-processing of captured images. OV9281 camera's module outlook is shown in Fig. 14.

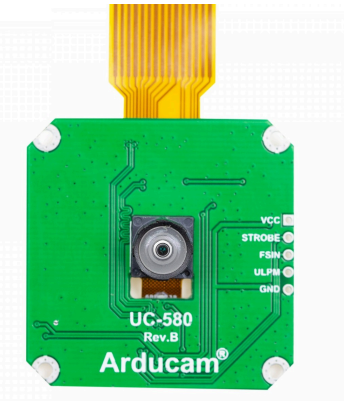


Figure 14: OV9281 camera's module outlook.

3.3.3 Display and Shield

The only available display driver in GAP library is ILI9341 adapted for Adafruit 2.8" TFT LCD Touchscreen. The display has 240x320 pixels with individual RGB pixel control, as well as a capacitive touchscreen attached to it. The display is compatible with 3.3V or 5V logic. It can be used in two modes: 8-bit and SPI. For 8-bit mode, 8 digital data lines and 4 or 5 digital control lines are needed to read and write to the display (12 lines total). Spi mode requires an SPI interface for data transfers (4 Pins: MOSI/MISO/CLK/CS), plus an I2C interface for control (2 pins: SCL,SCK) as well as 2 more pins for D/C control and power.

The 2.8" TFT touchscreen shield is assembled to the display (see Fig. 15). It is easier to use the shield with Arduino UNO, Mega, and GAP8 GAPUINO by just plugging in. However, the interfaces on GAP9_EVK are not designed to match the 2.8" TFT touchscreen shield to directly plug in. Therefore, in this project we just use some of the necessary pins from the shield to realize the core functionality of the display. We choose SPI mode to display logo, text message and images, connecting MOSI, MISO, SCLK, CS pins on MikroE bus embedded on GAP9_EVK and with PI_PAD_045 (GPIO_MIKROE_TX) configured as GPIO output to generate 3v3 D/C signal and GPIO_MIKROE_3V3 as power supply to the screen background light. We skip the I2C interface because the touchscreen functionality is not necessary for the game. The schematic of pin connection is shown in Fig. 16.

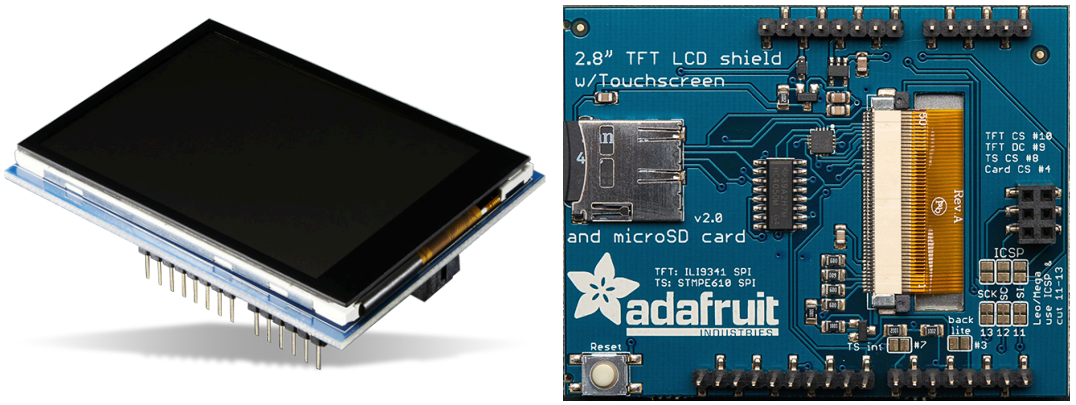


Figure 15: OV9281 module overview and functional block diagram

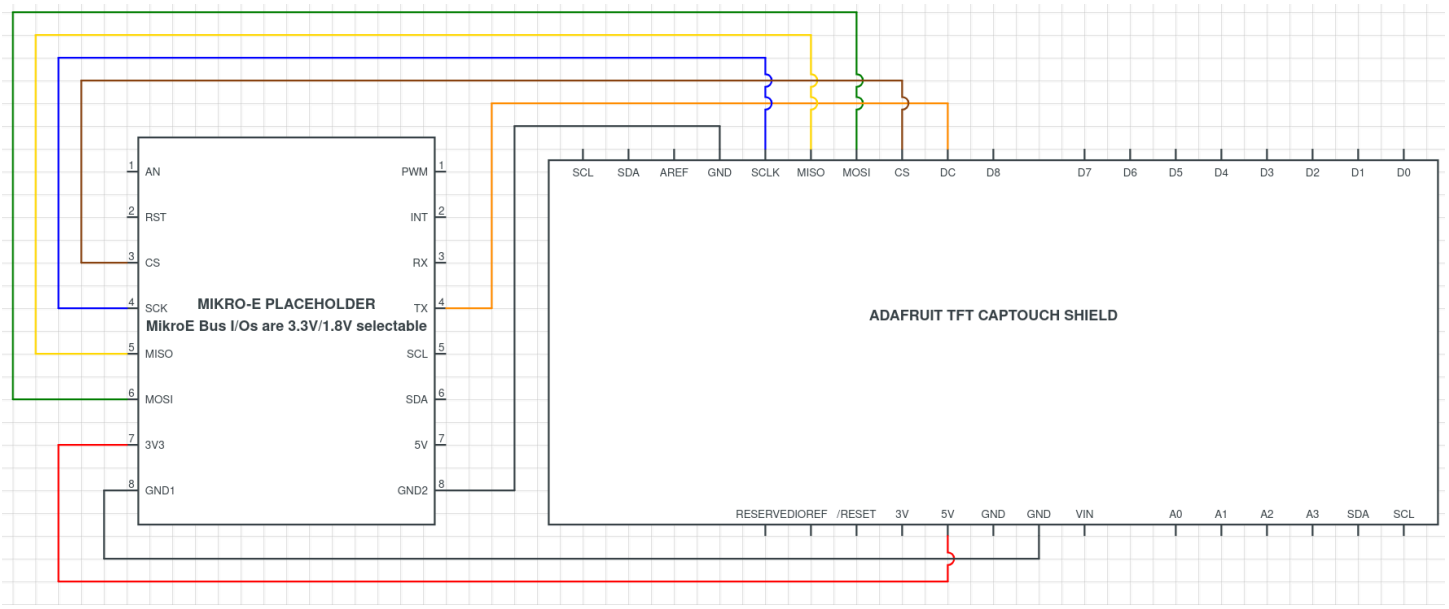


Figure 16: Schematic of pin connection between GAP9_EVK MikroE bus and Adafruit 2.8" TFT touchscreen shield

3.3.4 Audio Add-On board

GAP9's Audio Add-On board is an expansion board for GAP9_EVK designed to be stacked underneath it (see Fig. 17). The Audio Add-On board enhances GAP9_EVK with multiple audio-centric capabilities. It is intended to prototype portable audio applications. The Audio Add-On board enables audio capture into GAP9 through digital or analog microphones and ADC converter, as well as audio playback from GAP9 through audio DAC converter and headphone amplifiers.

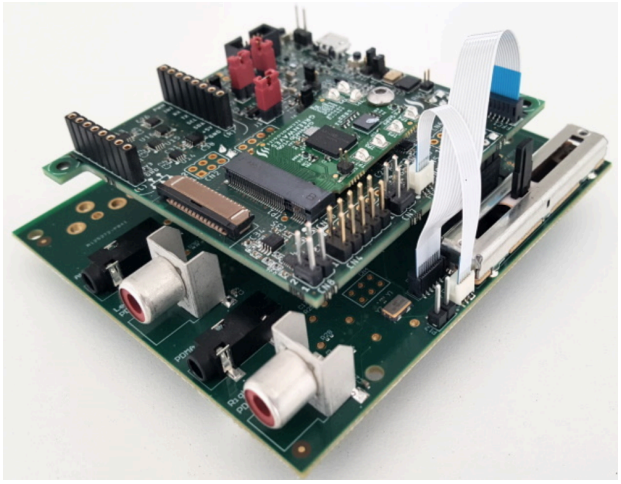


Figure 17: Audio Add-On stacked underneath GAP9_EVK, creating a complete system.

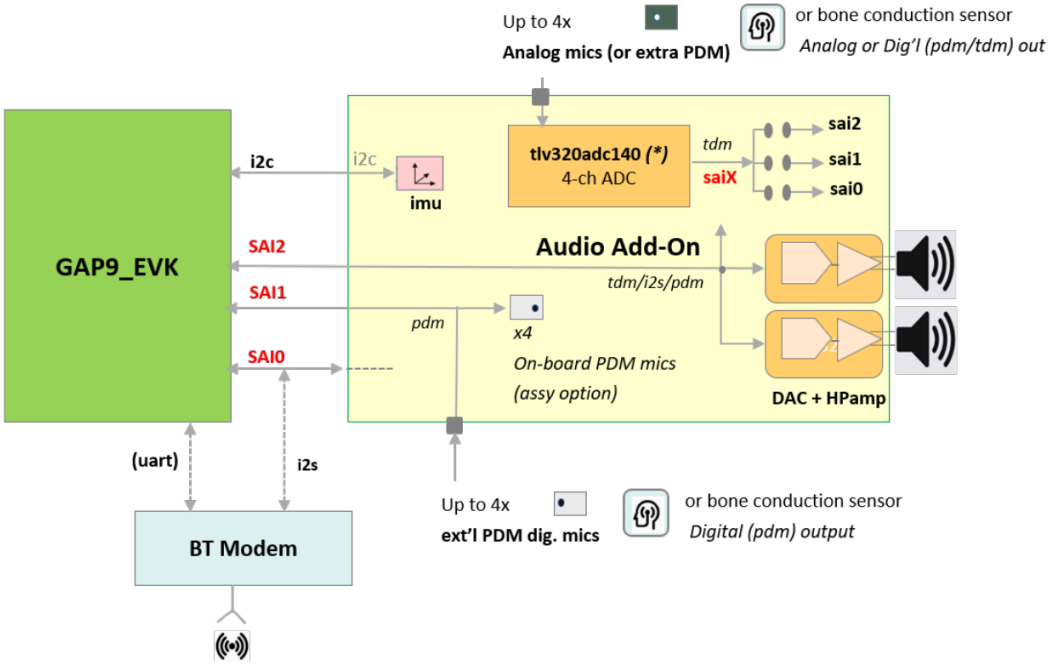


Figure 18: Audio Add-On / GAP9_EVK interfacing in a complete system

GAP9 provides 3 bidirectional Serial Audio Interfaces (SAI), as is shown in Fig. 18. Each SAI interface involves 4 signals: SAIx_SCK, SAIx_WS, SAIx_SDO, and SAIx_SDI. On the Audio Add-On board, the SAI2 interface is designed to receive digital audio from GAP9 in I2S, PDM format to go through A/D conversion and amplification to transmit on various connectors. Two alternative audio output paths are fed from SAI2 as listed below:

- **Path #1** is through the AK4332, a high-quality low power audio DAC with built-in headphone amplifier.
- **Path #2** is through an OpAmp-based 'Minimal Latency' PDM Amplifier.

In this project, to play the record which indicates moving phase to the players, we choose to use SAI2 through the AK4332 in PDM mode. The stereo signal is made available through a 3.5mm TRS jack located on Audio Add-On board, which can be connected to a speaker.

3.3.5 Servo Motor & Red Laser Pointer

We select servo motor SG90 (see Fig. 19) for the horizontal angle adjustment of the "gun", simulated by a 3.3V red laser pointer. The servo motor SG90 is tiny and lightweight with a relatively high output power. It can rotate approximately 180 degrees (90 in each direction). Parameters of servo motor SG90 are shown in Table 5.

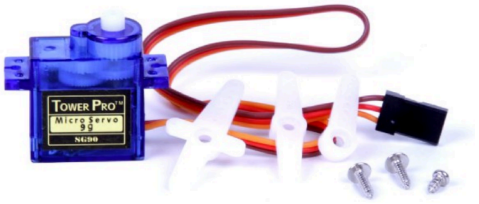


Figure 19: Servo motor SG90 overlook

Table 5: Servo motor SG90 parameters

Weight	11g
Rotation Angle	180°
Operating voltage	4.8-6 V
Torque	1,8 kg/cm (11 Ncm) (at 4,8 V); 2,4 kg/cm (15 Ncm) (at 6 V)
Speed	0,12 sec/60° (at 4,8 V); 0,10 sec/60° (at 6 V)
Control system	PWM (Pulse Width Modulation), Linear response to PWM for easy ramping
Pulse Frequency / Duty cycle	50 Hz / 20 ms square wave

The PWM (Pulse Width Modulation) signal is used to make the servo rotate forward or backward to a given position. The servo supports bidirectional rotation and the pulse duration of PWM signal determines the direction, as is shown in Fig. 20. The PWM signal period is 20ms (50Hz), and the servo motor angle ranges from 0° to 180°, linearly corresponding to the PWM signal pulse duration.

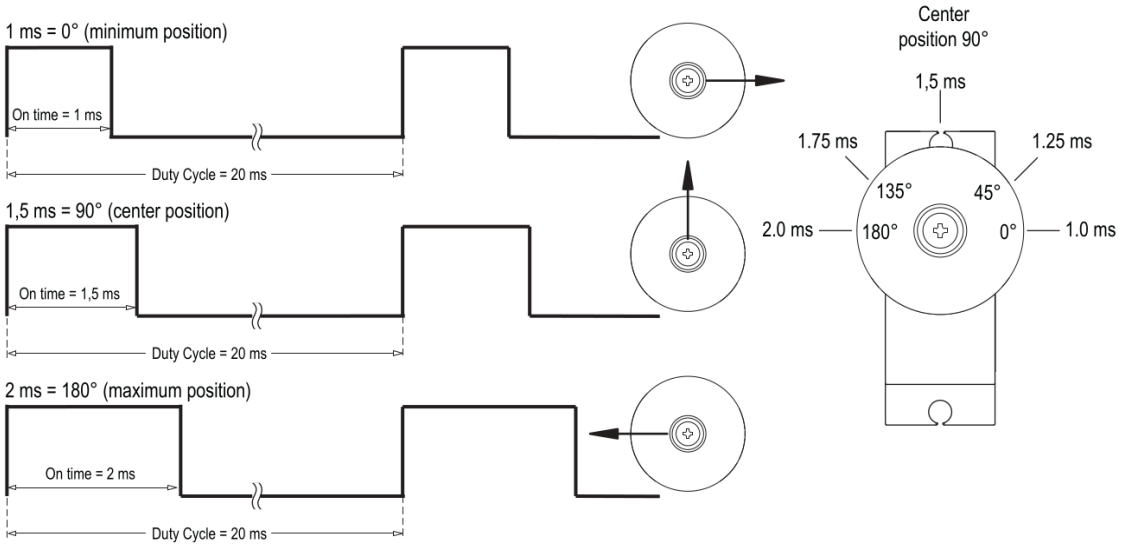


Figure 20: Pulse duration of PWM signal determines the direction [6].

Since GAP9_EVK's PWM signal generator is intended for high-frequency signals and it can only generate PWM signals of frequencies no less than 1000Hz, we need to use PI_PAD_067 (PWM pin on MikroE bus) as GPIO output to generate a PWM periodic signal for the control of the servo motor. The connection graph of GAP9_EVK MikroE socket and servo motor SG90 is shown in Fig. 21.

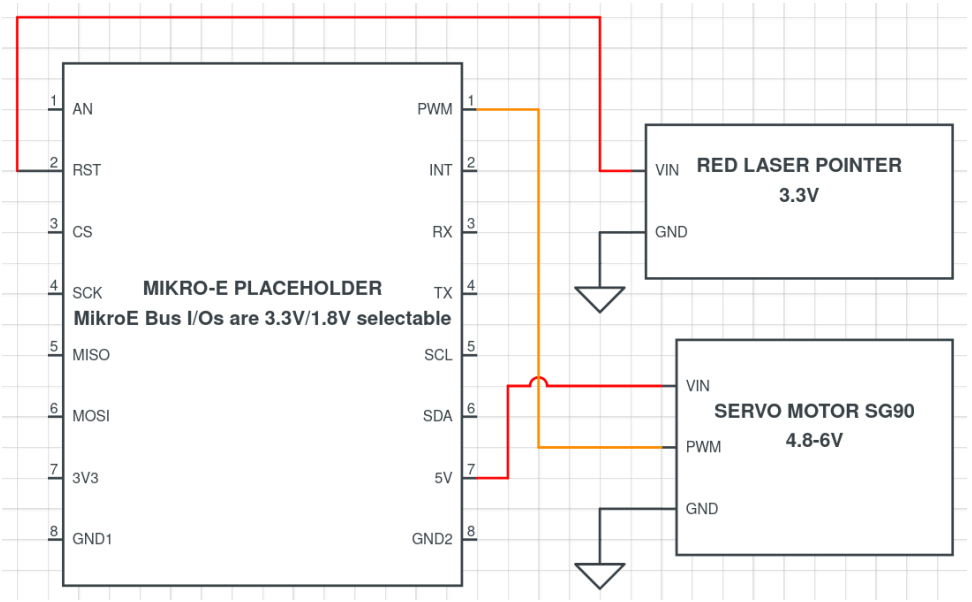


Figure 21: Schematic of pin connection between GAP9_EVK MikroE bus and servo motor SG90 and red laser pointer

The parameters of the 3.3V red laser pointer is shown in Table 8. For implementation, we use PI_PAD_086 (RST pin on MikroE socket) as GPIO output to control the on/off of the red laser pointer (see Fig. 26).

Table 8: Red laser pointer parameters

Transmit power	150mW
Life span	More than 1000 hours
Spot mode	Point light spot, continuous output
Laser wavelength	650nm (red)
Supply voltage	3V DC
Working current	<25mA

3.4 Schematic of Pin Connection and Hardware Assembly

The schematic of pin connection between MikroBUS socket on GAP9_EVK, Adafruit TFT display shield, servo motor SG90 and 3V3 red laser pointer is demonstrated in Fig. 22. To ensure a fixed relative position between the camera, servo motor and red laser pointer and to ensure a stable operation of the game system, we have designed a support shelf consisting of three parts tailored to the dimensions of all the selected hardwares during this internship. The 3D model has been built and converted to .stl model in CATIA, as shown in Fig. 23, and the slicing is made using the software Cura with the highest precision of 0.2mm. The parts are printed by the 3D printer

Anycubic Mega (see Fig. 24) with generic PLA material. Fig. 25 shows the hardwares assembled on the printed shelf and under deployment.

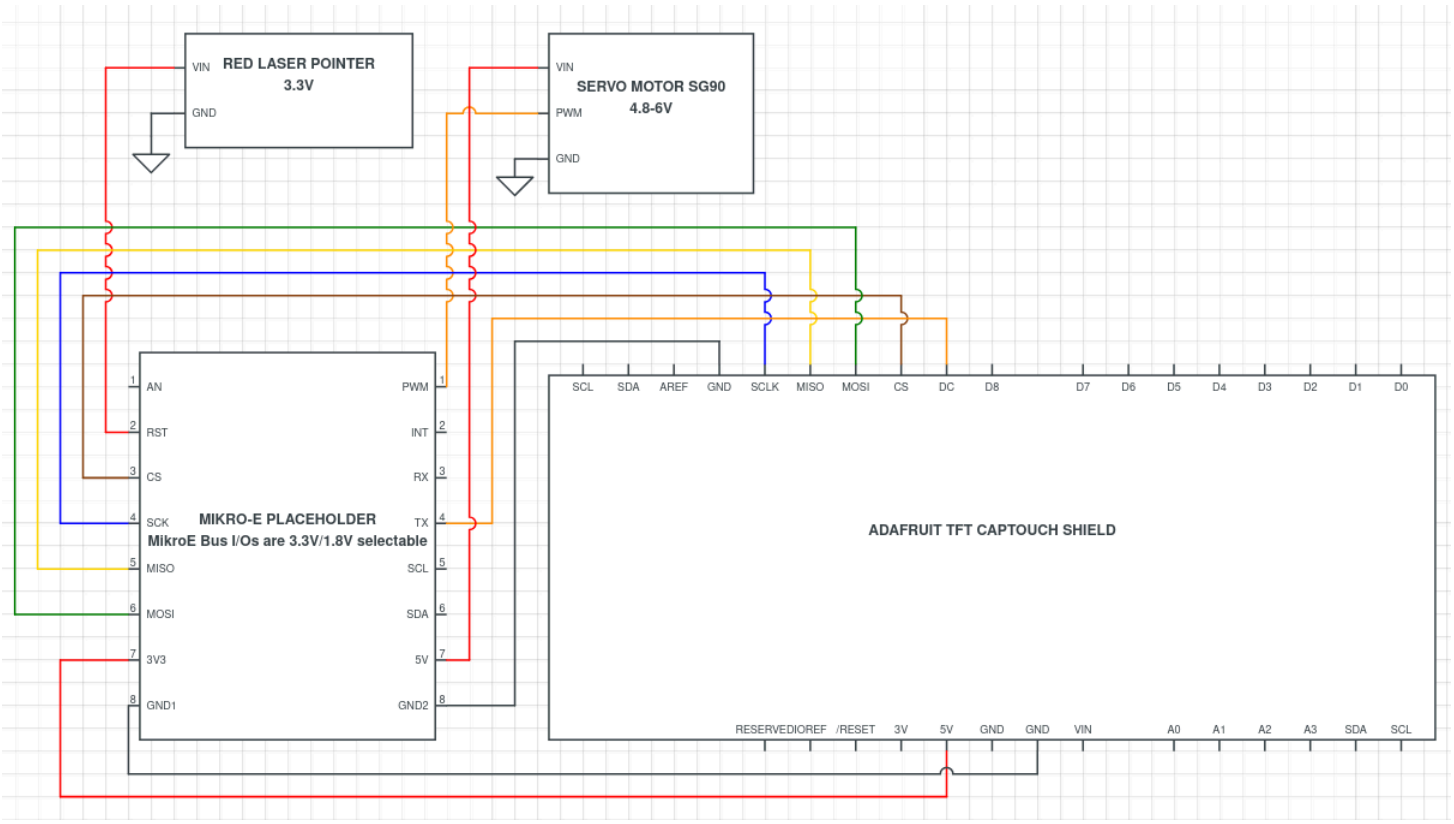
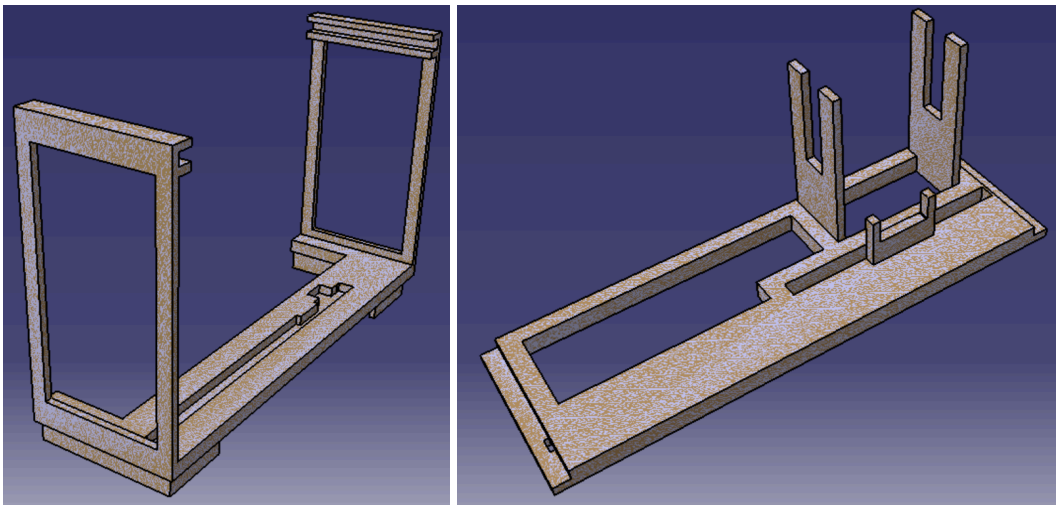


Figure 22: Schematic of pin connections between MikroBUS socket, 28” Adafruit TFT display, servo motor SG90 and 3V3 red laser pointer.



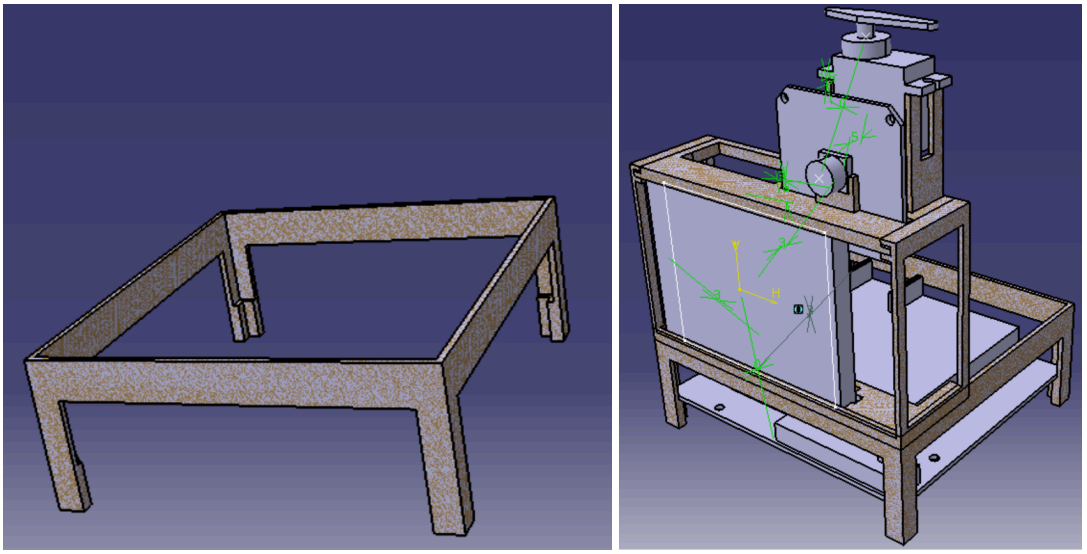


Figure 23: 3D model of the shelf designed for hardware assembly.

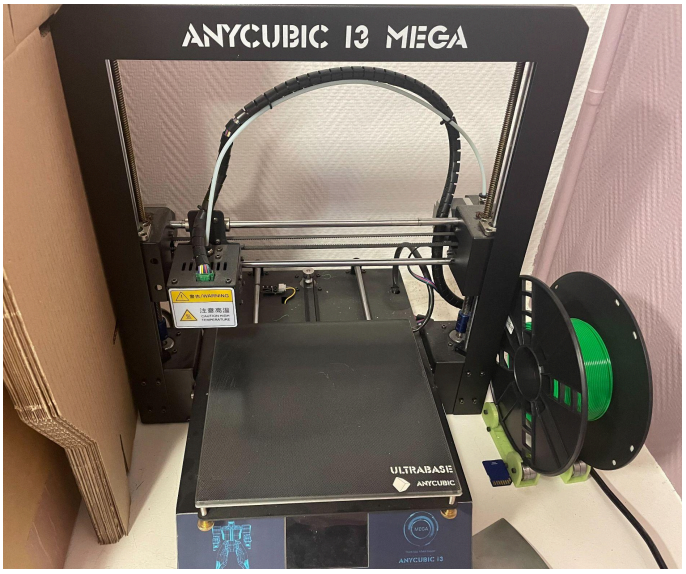


Figure 24: Anycubic Mega 3D printer.

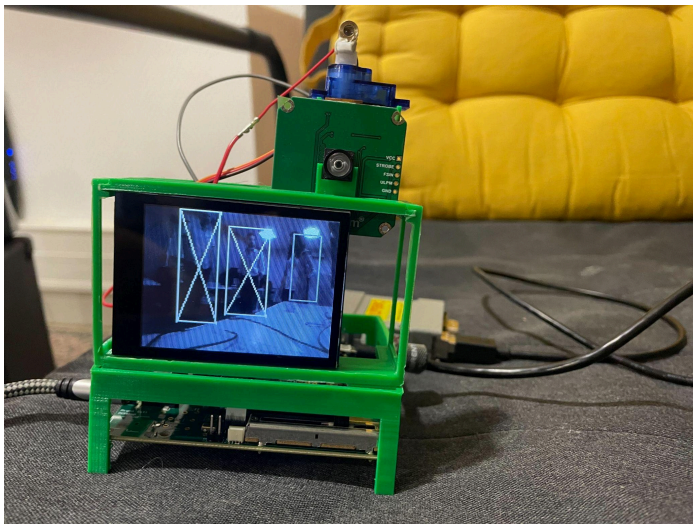


Figure 25: Assembled hardware on the printed shelf under deployment.

3.5 Software Integration

3.5.1 Memory Allocation

There are 4 types of memory on GAP9MOD:

- **Flash memory:** "dead" memory, used to store read only data, the data needs to be read and stored in L2 or L1 RAM to actually work on it,
- **L3 (hyper/octo) RAM:** temporary storage of read-write memory, but read/write operations are slow, the data also needs to be read and stored in L2 or L1 to work on it,
- **L2 RAM:** Main SoC RAM where the FC core works. It is relatively fast and is 1.5MB.
- **L1 RAM:** Cluster RAM where the cluster cores mostly work. It is the fastest RAM but is very small (128KB). It can communicate fast with L2 RAM via DMA (Direct Memory Access).

All functions transferring data between an external device and a chip memory must use the L2 memory for the chip memory. Since Autotiler has already generated the code to arrange L1 memory for real-time cluster computation of neural networks, the most critical memory left for user to arrange is the L2 memory. For the proper operation of software functions, we need to allocate two 1-channel image buffers of size 640x480, 1 byte per pixel. These buffers are designated to receive the grayscale images captured by OV9281 camera in 1-lane 8-bit output mode. Thus, two buffers of 250 Kb are allocated in L2 memory (see Equation (6)).

The original SSD model designed and trained by the AI team of Greenwaves is divided into 2 parts: CNN and SSD. The CNN model consists of convolution and max pooling layers, outputting features from 4 different layers in various scales. The SSD model is designed to take as input and interpret the output features (bounding box coordinates, classes and scores) of the SSD model, apply the softmax to the scores and the non-maximum suppression to bounding boxes.

In all, the CNN kernels consume 108Kb L1 memory, 200Kb L2 memory, 2.4Mb OspiRam L3 memory and 81Kb OspiFlash L3 memory. The SSD user kernels consume 38Kb L1 memory and 38Kb L2 Memory. Besides, a 12Kb bounding box buffer is allocated in L2 to store the output of 500 bounding box structures at maximum from the SSD model. For the real-time image transfer and storage on PC, we also allocate a 20Kb buffer for JPEG encoding. In the end, the game sound record that indicates the moving period takes 91Kb L2 buffer. The memory consumption of the critical L2 memory is calculated as below:

$$\begin{aligned} L2\ memory\ consumption\ &= 2 \times 250KB(image\ capture\ buffer) + 200KB(CNN\ buffer) \\ &+ 38KB(SSD\ buffer) + 12KB(bounding\ box\ buffer) \\ &+ 20KB(JPEG\ buffer) + 91KB(sound\ record) \\ &= 861\ KB < 1.5MB \end{aligned} \quad (6)$$

3.5.2 Reference of CNN and SSD Models

The NN model has been transformed from body_detection.tflite model into application code automatically through NNTool and Autotiler tools. In the main function, we just need to allocate memory and construct the NN graph, and then refer to the functions RunNN() and RunSSD() in sequence during the game loop. Note that the outputs extracted from 8 feature layers of the NN model are fed into the SSD model as input, thus for a safety operation, we have to use synchronous functions to block FC execution and wait for the completion of CNN and SSD models reference, as shown in the pseudo-code below:

Pseudo-code of CNN and SSD models reference

```
Initialization, memory allocation and hardware configuration
Construct and initialize CNN and SSD models
Start game_timer //game starts
while game_timer < GAME_TIME do
        Play game sound //moving phase starts
        Wait for game sound ends // moving phase ends
        Reset still_phase_timer //still phase starts
        while still_phase_timer < random_period do
                ...
                Prepare CNN task and send to cluster //run CNN model
                Wait for the finish of CNN task
                Prepare SSD task and send to cluster //run SSD model for post-processing
                Wait for the finish of SSD task
        end while
end while
```

The application code of main function can be found in **Appendix 2**. To run the CNN and SSD models on the 9 clusters which are designed for parallel computation tasks, we use the APIs pi_cluster_task(), pi_cluster_send_task_to_cl_async() and pi_task_wait_on() to prepare and initialize a cluster task for execution, enqueue asynchronously a task for execution on the cluster, and block the execution of FC until the specified notification event created with pi_task_block() has been triggered. In the functions RunNN() and RunSSD(), we use the APIs gap_cl_readhwtimer() and pi_time_get_us() to count the cycles on hardware (i.e. GAP9 clusters) and the model execution time in microsecond. It is a common method that we use to evaluate the performance of neural networks as well as peripherals in terms of execution speed on GAP9_EVK.

We have tested the computation time and cycles running CNN and SSD models both on GVSOC and on GAP9_EVK and recorded that the computation time for referencing CNN and SSD is around 108ms (42206478 cycles) and 10ms (3970124 cycles) respectively. In total, it takes only less than 120ms to process a frame, which is ideal for real-time operation.

3.5.3 CSI-2 MIPI Interface

In this project, We use the OV9281 camera because it provides 8-bit grayscale images to the master device via CSI-2 MIPI interface, which is compatible to the camera interface on GAP9_EVK. We note that the pixel clock frequency supported by OV9281 is around 25MHz, which is far too low for other peripheral devices. In order to maintain the high performance of GAP9 peripherals, we keep the peripheral clock frequency equal to 400MHz, the maximum peripheral frequency supported and guaranteed by GAP9_EVK. To fix the pixel clock frequency of the OV9281 camera to 25MHz, we use a clock divider in the camera driver which autonomously divides the peripheral clock to adapt to the required pixel clock frequency.

The CSI-2 connector implemented on GAP9_EVK follows the standard pinout of the 15-pin FFC cable employed by RPI cameras. Those cameras typically use IO-0 of the connector as Power

Enable to the camera module. On GAP9_EVK, IO-0 of the connector is controlled through the GPIO expander FXL6408UMX, controlled from GAP9 through the I2C3 bus. Following is an example sequence to access such a camera from GAP9:

- Make sure J1 is fitted on GAP9_EVK (so 3V3_CAM is propagated across the board)
- Set GPIO0 of GAP9 to Logic-1 to enable the on-board low-dropout regulator (LDO) delivering 3V3_CAM. Wait at least 550us before any subsequent access to camera or GPIO expander to leave enough time for LDO output to stabilize at 3.3V.
- Set up I2C3 (Alternate 3) on GAP9 to access the GPIO expander (I2C address=0x44 expressed on 7 bits, excluding R/W flag MSB) and CSI-2 camera.
- Set IO-0 of the GPIO expander to Logic-1 via suitable I2C accesses (this sets CAM_IO-0 input of CSI-2 camera, i.e. asserts Power Enable input of camera)

The camera configuration settings are set up during the software initialization. During the game loop, a command is sent to the camera to control the start and stop of image transmission continuously during the still phase in order to capture the players' motion in real-time. To capture images rapidly during the still phase, the camera is always on to reduce the capture latency.

With the above-mentioned camera control scheme, we test the capture time on GAP9_EVK connected to OV9281 camera via MIPI CSI-2 bus. Working at 25MHz pixel clock, the camera reaches around 28ms per capture, which meets our requirement of real-time capture.

3.5.3 SAI2 AK4332 Interface

The AK4332 is a high-quality, low-power 32-bit Mono DAC with an integrated headphone amplifier and volume control. It is configured through I2C and accepts PCM data over an I2S protocol, as well as 1-bit sigma-delta data in PDM or DSD64 format.

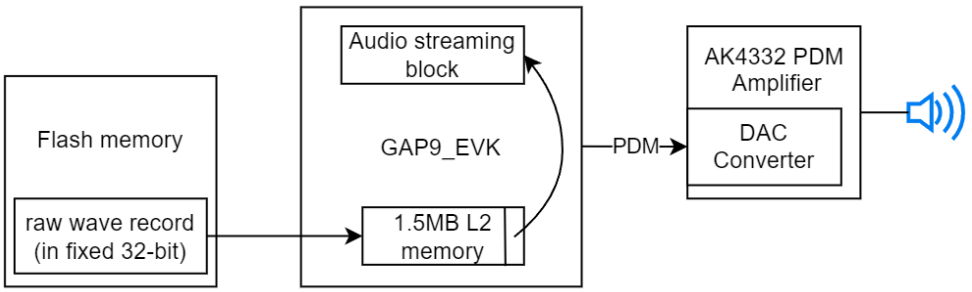


Figure 26: The pipeline of input record conversion, streaming and amplification.

In this project, the game sound record is stored in 32-bit fixed-point format in a pre-registered raw file named raw_wave.h. It is played via continuous streaming on a speaker connected to the TRS jack located on Audio Add-On board through the amplifier AK4332 in PDM/DSD mode. Resampling and conversion to PDM signal is done via a hardware block embedded on GAP9 that allows streaming audio with low latency and low power. Fig. 26 demonstrates the pipeline of input record conversion, streaming and amplification.

In this project, we use the AK4332 in PDM/DSD mode taking an external master clock as input reference, which comes from the 12.288MHZ/24.566MHz clock oscillator implemented on the Audio Add-On board. The PDM clock is expected to be 64 times the audio sample rate, i.e. 3.072MHZ for 48KHz (the maximum supported sample rate). The detailed power-up sequence of

AK4332 is described in **Appendix 3**. Note that the power-down sequence of AK4332 is not considered in this example.

After the initialization and power-up of AK4332 PDM output channel, we use the API function pi_i2s_ioctl() to control the start and stop of data transmission through I2S (Inter-IC Sound) interface. During the data transmission through the I2S interface, we use pi_yield() to stop the other processes in FC and wait until the end of transmission. When the transmission is completed (notified by a callback function), pi_i2s_ioctl() is called again to stop the I2S interface. Afterwards, the input wave record goes through pre-processing and resampling procedures again to prepare for the next I2S transfer. The set-ups to initiate and power up AK4332 and to control I2S interface to play the record are written in function ak4332_init() and ak4332_play() in game_sound.c respectively, and the latter is referred to repeatedly in each moving phase of the game.

The frame clock frequency is set to $307200 \times speed$, which is updated every time when the I2S interface is reopened, with an increasing play speed ranging from 1.0x to 10.0x (corresponding to the maximum PDM sample frequency of 3.072MHz).

The limitations of the game sound playing through AK4332 amplifier PDM mode lie in that only one PDM channel (either left or right) on SAI2 bus can be enabled on the Audio Add-on board, and that simply increasing the frame clock frequency (I2S clock) results in not only the speed up of the sound record but also a higher tune. Signal processing techniques are needed to maintain the same tune of sound when the play speed increases, but for the moment we put aside the sound quality since it is beyond the scopet of this project.

3.5.4 MikroBUS Socket and Other Peripherals

The MikroBUS socket is intended to plug MikroE Click board. It consists of 3 groups of communication pins (SPI, UART, I2C), 6 additional pins (PWM, Interrupt, Analog Input, Reset and Chip Select) and power supplies. All pins of the mikroBUS except the analog input pin AN are connected to GAP9 I/Os as per the mapping described in **Appendix 1**.

3.5.4.1 ILI9341 TFT display - SPI interface

The connection between the ILI9341 Adafruit TFT display and GAP9_EVK is via SPI bus on the MikroBUS socket, as shown in the schematic (see Fig. 16) in **Section 3.3.3**. The concerned pins are listed in Table 9 below. It is necessary to choose Alternate0 of the pin function to activate SPI bus. In addition, DC3V3 is connected to the TX pin configured as GPIO output (Alternate1) to signify control commands. When DC3V3 is pulled up to Logic-1, the slave (ILI9341 display) is ready to receive any control command from the master (GAP9_EVK).

Table 9: Pin connection and pin function selection on MikroE socket for ILI9341 Adafruit display control through SPI bus

Pins on ILI9341 display (slave)	Pins on GAP9_EVK (master)	Pad ID of pins on GAP9_EVK	Selection of pin function on GAP9_EVK
CS	CS0	PI_PAD_034	Alternate0
MISO	MISO	PI_PAD_039	Alternate0
MOSI	MOSI	PI_PAD_038	Alternate0
SCK	SCK	PI_PAD_033	Alternate0

DC3V3	TX (GPIO)	PI_PAD_045	Alternate1
POWER	3V3	PI_PAD_035	Alternate0

Before configuring the MikroE socket, it is important to enable the MikroE bus by setting the 3V3 pin (PI_PAD_035) to GPIO output mode and setting it to Logic-1, as done in the function enable_3v3_periph(). Since the ILI9341 TFT display driver is adapted for the use on GAP8 in GAP library it did not support implementation on GAP9 chips. During this internship we tested and added the settings in gap9_evk.c and gap9_evk.h, which takes care of the configuration and initialization of the SPI bus connecting GAP9_EVK and ILI9341 TFT display such that the ILI9341 driver is ported to GAP9. In the main function of the game system, we refer to the APIs pi_display_open() to and pi_display_ioctl() to initialize and turn on the ILI9321 TFT display.

Afterwards, several API functions in ili9341.c such as setCursor(), writeLogo(), writeTriangle(), writeCircle(), writeRect(), setTextColor() and writeText() are called to draw the game logo and the text “SQUIDxGAME”, as well as the Greenwaves’ logo and the text “Greenwaves Technologies” in predefined colors on the display, through sending data of pixels in bytes via the SPI interface.

In the moving phase of the game loop, we display a black background with the white text “RUN!!!”; while in the still phase, the camera continuously takes photos while GAP9_EVK processes the images in real-time and sends the post-processed images to the display via SPI interface. To send the image, we simply need to prepare a buffer and set the buffer’s pointer to the memory location of the image data buffer, which is resized in advance to adapt to the display size of 320x240 pixels.

We have tested the performance on hardware and recorded that the time required to send an image to the ILI9341 display is around 28ms at the GAP9 peripheral frequency of 400MHz.

3.5.4.2 SG90 Servo Motor & Red Laser Pointer - PWM Signal Generation

Since the PWM generator embedded on GAP9 is intended for high-frequency clock generation and it can only generate PWM signals of $frequency \geq 1000Hz$, in this internship we have to generate a PWM signal manually in software in order to meet the requirement of the SG90 servo motor - a PWM signal of 50Hz (period=20ms) - to control the output shaft angle. We choose PWM pin (PI_PAD_067) and SDA pin (PI_PAD_086) on MikroE bus to connect to the PWM input signal of the SG90 servo motor and VCC power supply of the red laser pointer respectively.

All functions on the FC side are by default synchronous and block the caller until the operation is done. In order to generate the PWM signal to control the output shaft angle of SG90 in parallel with the other functions and computations (i.e. no interruption to other threads running in FC), we use the APIs pi_task_push() and pi_task_push_delayed_us() which are intended for generating asynchronous tasks in the same core on GAP9. Specifically, the APIs pi_task_push() and pi_task_push_delayed_us() are used to trigger the specified notification (after the specified delay for the latter), which will schedule the callback execution.

To realise the function of PWM signal generation, we define a structure type named task_servo, which contains all the pi_tasks necessary to generate the PWM signal and to control the red laser pointer, as well as required arguments (x_pos, active_time_us, etc.) for the callback functions. Then in the function task_servo_init(), we associate the tasks with their corresponding callback functions.

The task_servo structure itself is fed into the callback functions as an argument, in order to easily refer to the other tasks and the parameters in the callback functions.

In the task_servo structure, the most foundational pi_task which is kicked off in the first place is mytask. As shown in Fig. 27, the callback_function() associated with mytask pushes two subtasks, namely mytask1 and mytask3, which respectively take charge of generating periodic PWM signal and turning on the red laser pointer at the proper moment, i.e. with a delay of 1s waiting for the servo output shaft to stabilize to the intended position. The callback_function1() of mytask1 sets Logic-1 to the PWM pin (PI_PAD_067) for a certain active time before pushing mytask2. The callback_function2(), associated with mytask2, sets Logic-0 to the PWM pin (PI_PAD_067) for $(20ms - active\_time)$ to complete the PWM period of 20ms. Then the callback_function2 pushes mytask1 again to generate the PWM signal periodically. In the meanwhile, after a delay of 1ms, mytask3 is pushed from callback_function(), with its callback_function3() setting Logic-1 to the SDA pin for 2s (simulate shooting of the gun) before kicking off mytask4, and the callback_funtion4() associated to mytask4 is intended to set the SDA pin back to Logic-0. Mytask5 and callback_function5() are designed to update the active time of PWM signals after a delay of minimum_last_time using the formula in Equation (7), which converts the intended shoot position into PWM active time.

$$PWM\_active\_time = 2100 - 6.25x\ (us) \tag{7}$$

When there is a wait list of x positions to shoot, we make a loop to take these positions in turn in the callback_function() and arrange future tasks to be kicked off according to the number of x positions and the minimum last time of each PWM signal pattern. Finally, mytask6 and callback_function6() are used to signify the completion of all the servo tasks in the x position list. Note that in order not to interrupt the on-going servo task on the hardware, we simply avoid pushing the new servo tasks and skip the current frame when the servo_task_fini signal is not equal to 1. Details of the application code concerning the pushed tasks and callback functions can be found in **Appendix 4**.

Equation (7) is deduced from our experiment with the OV9281 grayscale camera, servo and red laser pointer fixed in the same vertical axis (see. Fig 28). Fig. 29 shows the positions of the red laser at increasing PWM active time from 1.1ms to 2.1ms at 50Hz frequency, with an increment of 0.1ms. From the captured images, we can measure that the PWM active time from 1.1 to 2.1 corresponds to the horizontal axis from 160 to 0 with a linear relationship, as indicated in the SG90 servo datasheet. Therefore, we deduced Equation (7) to convert the intended x position into PWM active time.

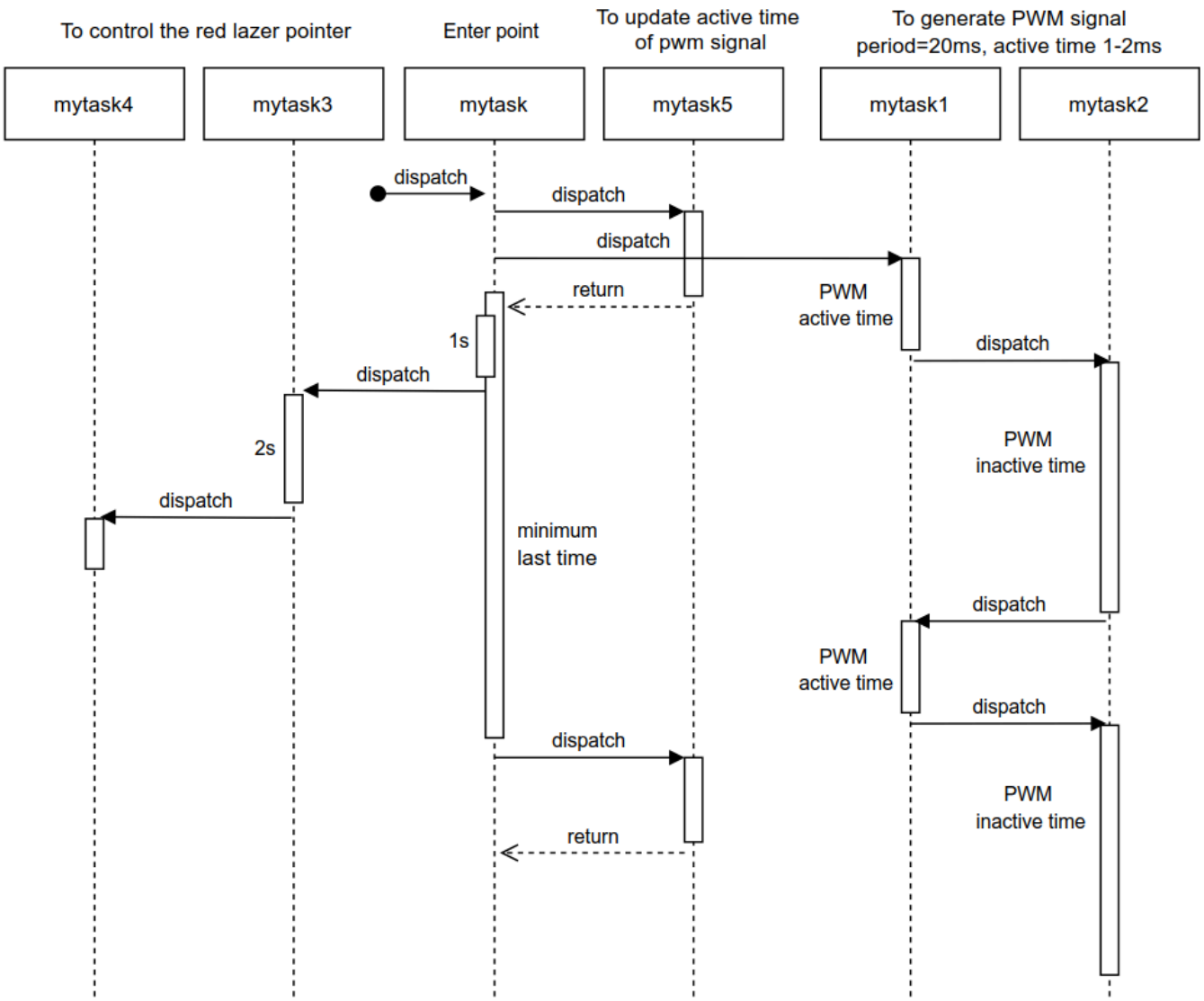


Figure 27: Calling sequence of parallel servo tasks for PWM signal generation

Figure 28: Fixed positions of OV9281 grayscale camera, servo and red laser pointer in the same vertical axis in the experiment.

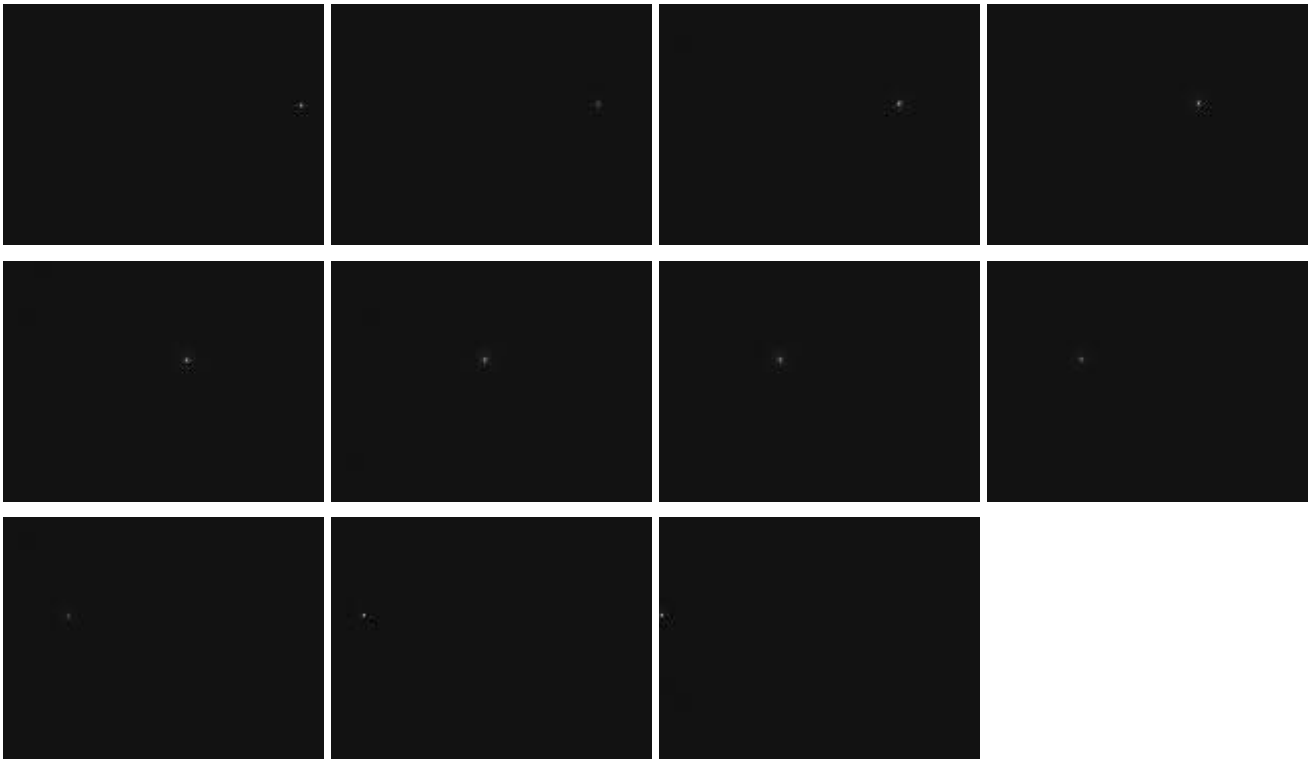


Figure 29: The positions of the red laser at increasing PWM active time from 1.1ms to 2.1ms at 50Hz frequency, with an increment of 0.1ms.

In the game loop, we call the shootLaserPointer() function (see the pseudo-code below) after each frame differencing calculation. This function iterates through all the bounding boxes with any movement detected (i.e. bbox_alive=2). Then the x positions of the center of these bounding boxes are calculated as the targets of shooting. The x positions are sent to the servo_task structure as a parameter to be processed in the callback_function().

Pseudo-code of shootLaserPointer function

```
if previous servo task is finished then
        Collect all the x positions to shoot the gun
        for counter = 0 to number_of_bounding_boxes
                if bbox_alive = 2 then
                        Calculate x position of the bounding box center
                        x_pos[id] ← x_position
                        id++
                        if id=MAX_SHOOT_POS then
                                break
                        end if
                end if
        end for
        if id > 0 then
                Initiate servo task
                Push servo task
```

end if
end if

The functions above are intended to operate the SG90 servo motor and the red laser pointer in parallel with real-time SSD and frame differencing calculations. However, during the game operation it has some limitations:

- The shooting algorithm only takes some of the frames into consideration, i.e. every 3 seconds after the completion of the previous servo task. The movement of players in the other frames is ignored.
- After the fake shooting with the red laser pointer, the players are actually not ruled out from the game system and will still be taken into account in the following frames. Thus if a player keeps on moving during the still phase, the servo motor will still count the player as a valid and "alive" player and control the red laser pointer to track the player continuously.

3.5.5 Test Mode Functions & Mode Selection

To test and verify the feasibility of the game algorithms on PC, we use GVSOC to simulate the the operation on GAP9 from PC, using the APIs to read images from the PC, encode into JPEG and send images to PC via the file system to replace the functionality of camera and display. The functions using FS (File System) and JPEG encoder libraries to read and send images to the PC are written in jpeg_transfer.c. We choose to compress the image to JPEG format in order to reduce image size and thus to improve the transfer speed.

To enable the test mode and inspect the simulation of the game system on the PC, the user can choose to uncomment TEST_MODE=1 in Makefile. When TEST_MODE=1 is defined, the program will block the usage of audio amplifier, camera, display, servo motor and red laser pointer, instead, it will set SEND_TO_LAPTOP=1 which allows the encoding to JPEG and sending processed images to the PC through the file system. Since the camera is blocked, the game system will read images from the PC in a user-defined folder as input images.

3.5.6 General Game Sequence

The general game sequence, as described in the sequence diagram of the "1-2-3 GAP" game system in **Section 2.1** (see Fig. 8), is written in the main function start() in main.c, as shown in the pseudo-code below. Application code can be found in **Appendix 2**. It is composed of three parts:

- Initialization, memory allocation and hardware configuration
- Game loop with iterating moving and still phases
- Free dynamic memory and close the cluster

Main function of the game system

Set the frequencies: fc_freq ← 400MHz, cluster_freq ← 400MHz, peripheral_freq ← 400MHz
Memory allocation and hardware configuration.
Initiation and memory allocation of CNN and SSD graphs
Start game_timer //game starts
while game_timer < GAME_TIME **do**
    if SEND_TO_SCREEN **then**
        Display the text "RUN!!!" on the TFT screen

```
        end if
        if GAME_SOUND then
                Play game sound //moving phase starts
                Wait for game sound ends // moving phase ends
        end if
        Reset still_phase_timer //still phase starts
        while still_phase_timer < random_period do
                if FROM_CAMERA then
                        Capture image and send to controller
                else
                        Read image from PC
                end if
                Resize the image to 160x120
                Calculate frame difference and detect moving players of the previous frame
                Draw bounding boxes on the previous frame
                if SEND_TO_SCREEN then
                        Send the previous frame to the screen
                else
                        Encode the previous frame to JPEG and send to PC
                end if
                if SERVO_MOTOR then
                        Send asynchronous servo task to the servo motor and red laser pointer
                end if
                Prepare CNN task and send to cluster //run CNN model
                Wait for the finish of CNN task
                Prepare SSD task and send to cluster //run SSD model for post-processing
                Wait for the finish of SSD task
        end while
end while
//game ends
if SEND_TO_SCREEN then
        Display the text “GAME OVER!!!” on the TFT screen
end if
Free CNN and SSD memory and close the cluster
```

All the functions deployed in the game system are synchronous because of their dependency on one another, except the shootLaserPointer() function which drives in parallel the servo motor and red laser pointer with a physical delay. In total, during the still phase when the game system continuously detects moving players, it reaches 5fps for feasible real-time operation on hardware.

4 Results and analysis

4.1 Game Demo

To play the game "1-2-3 GAP", the user needs to connect GAP9_EVK to the PC via a USB cable or through OLIMEX-OCD JTAG debugger. It is also possible to flash the program into GAP9's L3 memory for the boot and supply the board with 5V external power.

After connecting GAP9_EVK to the PC, the user should first open a new terminal and then use the command `$ source sourceme.sh` in the gap_sdk folder on PC and choose option2 - GAP9_EVK _AUDIO to run the game on board or option3 - GAP9_V2 to simulate the game on GVSOC. If the communication and file transmission between GAP9_EVK and PC is realized through the OLIMEX-OCD JTAG debugger, the user will have to use the command `$ export GAPY_OPENOCD _CABLE=interface/ftdi/olimex-arm-usb-ocd-h.cfg` before running the program. After the hardware selection, the user should go to the directory nnmenu/starters/squid_game/1-2-3_GAP/ and use the command `$ make all platform=board && make run platform=board` to run the program on GAP9_EVK, or alternatively use the command `$ make all platform=gvsoc && make run platform=gvsoc` to simulate the game sequence on GVSOC.

The user can choose several options in Makefile as listed below:

```
## To simulate the game sequence on GVSOC platform
# TEST_MODE=1
## Mute printf in source code
SILENT=1
## Enable image grub from camera
FROM_CAMERA=1
## Enable display on TFT screen
SEND_TO_SCREEN=1
## Enable send to PC
# SEND_TO_LAPTOP=1
## Enable audio amplifier on Audio Add-on board
GAME_SOUND=1
## Enable peripherals (servo motor and red laser pointer) on MikroE bus GPIO pins
SERVO_CONTROL=1
```

The user can simply comment or uncomment the options to choose from the peripherals to be deployed, i.e. either use the camera or read images from file, either display infomation and images on Adafruit TFT display or not, either play the game sound record via AK4332 amplifier or not, etc. Note that when the user chooses TEST_MODE, all peripherals are disabled to simulate the game algorithm with GVSOC platform on PC.

Two parameters of the game system can be defined by the user in main.c (see the code below), i.e. SERVO_WAIT_TIME and GAME_TIME, which determine the predefined operation time reserved for each "shoot" and the time limit of the game. When the GAME_TIME is used up, the game loop ends, all the players still detected inside the FoV of the camera will be ruled out and the text "GAME OVER!!!" will be displayed on the screen.

Fig. 30 shows the messages printed on the command window during the game loop, with options FROM_CAMERA=1 and SEND_TO_LAPTOP=1. The demo video of two players playing the game can be found in the file demo.mp4.

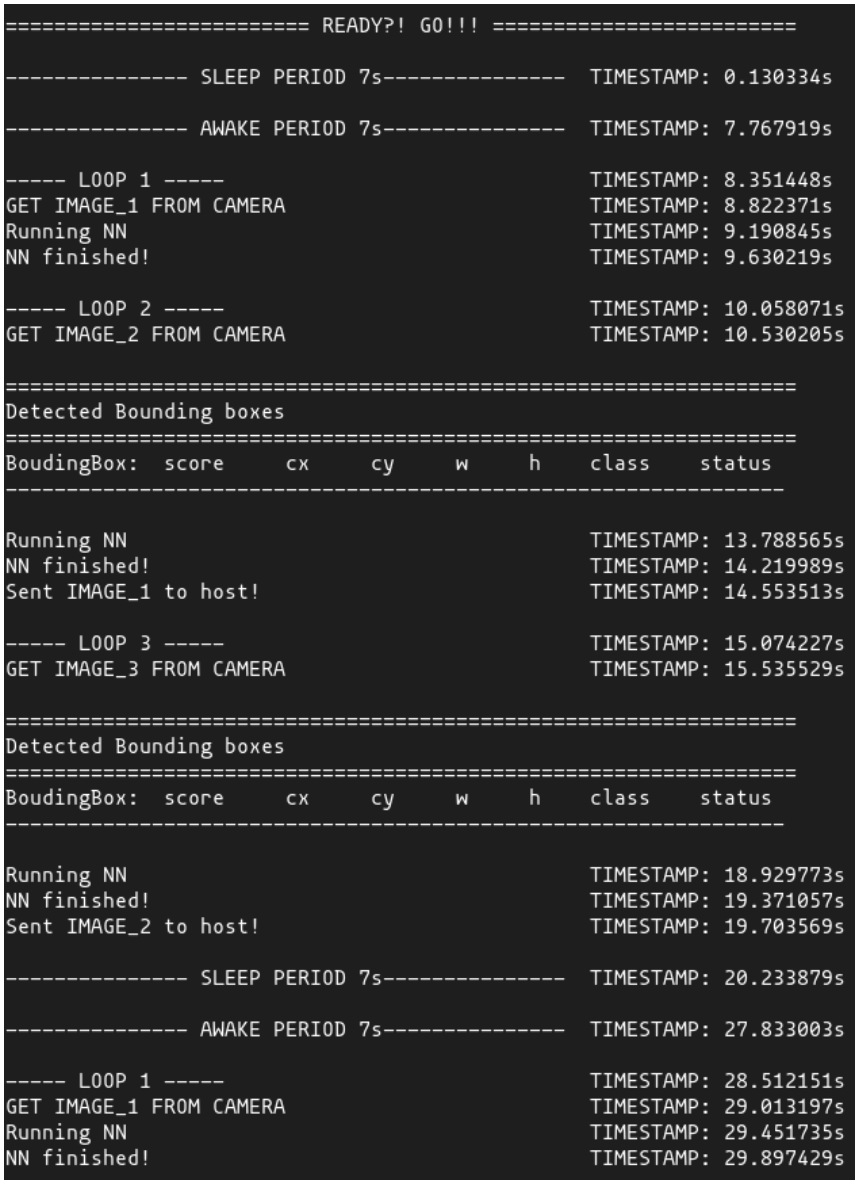


Figure 30: The messages printed on the command window during the game loop, with options FROM_CAMERA=1 and SEND_TO_LAPTOP=1.

4.2 Performance

4.2.1 Computation & Transmission Time

During the internship, we have first tested and recorded the computation time of the SSD and YOLOX models on GAP board and GVSOC as a reference. The results are shown in Table 10. The body detection SSD model, which is under deployment on GAPUINO8 body detection application, runs at a reference speed of 4.2fps which realises real-time inference. In the test, the new generation of GAP9 has proved to run the model 2 times faster than GAP8, reaching 8.1fps and 7.6fps on GVSOC and on board respectively. This is due to higher FC maximum frequency and larger memory space. For comparison, we have quantized a people detection model from the YOLOX family based on the backbone of yolox-tiny trained by Xperience AI and tested its inference time on GVSOC. The model takes 676ms to process a frame, resulting in a reference speed of 1.5 fps when we choose to use NE16 (a new feature of GAP9, basically a core dedicated to NN reference) during quantization. If we do not choose to use NE16, the YOLOX model runs at a much slower speed, namely only 0.4fps. To conclude, despite higher accuracy, the yolox_tiny model is not suitable for real-time inference on GAP9 because of its relatively slower reference speed.

Table 10: Inference speed of SSD and YOLOX models on GAP board and GVSOC

Model name	Quantization	Input size	Input format	Platform	Model reference time (ms)	Total cycles	FC frequency (MHz)	FPS
Body detection SSD	Scale8	160x120	Grayscale	GAP8 - GVSOC	317	55412700	250	4.5
				GAP8 - board	340	59440856	250	4.2
				GAP9 - GVSOC	123	49413727	400	8.1
				GAP9 - board	131	52600393	400	7.6
Yolox tiny	Scale8 NE16	640x480	Raw Bayer	GAP9 - GVSOC	676	249024304	400	1.5
	Scale8			GAP9 - GVSOC	2778	1005306316	400	0.4

During the game deployment, we have recorded the time consumption for the main functions of the game system on hardware, including transmission of the captured images from OV9281 camera to GAP9_EVK, CNN and SSD model reference, frame difference calculation, encode to JPEG format, transmission of the images to the laptop and to Adafruit TFT display. The test has been carried out at $FC\ frequency = Cluster\ Frequency = Peripheral\ frequency = 400MHz$, and the results are listed in Table 11. From the table, we can find that the most time-consuming process of the game system is the transmission of images from GAP9 to PC through the file system. This gives us the idea of porting the Adafruit TFT display to GAP9 instead of sending images to PC, and if sending to PC is inevitable, of making the image transfer an asynchronous process which runs in parallel with NN model reference to reduce total computation time.

Table 11: Computation & transmission time of SSD model inference and interaction with peripherals on GAP9_EVK

Activity	Computation time (ms)	Cycles	FPS	Annotation
Capture & send image from OV9281 camera to GAP9_EVK	28	45783740	35.7	-
Calculate frame difference	0	1167	-	-
Reference CNN model	108	42206478	9.3	-
Reference SSD model	10	3970124	100	Take the output of CNN as input
Encode images to JPEG format*	5	1978624	200	To save image transmission time
Send images to laptop via OLIMEX-OCD JTAG debugger*	497	198859286	2	Asynchronous process in parallel with CNN and SSD model reference
Send images to Adafruit TFT display	28	11216013	35.7	-

Total	<120	-	>5	The encoding and transmission of images to laptop is disabled

In total, during the still phase when the game system continuously detects moving players, it reaches more than 5fps, which is feasible for real-time operation on hardware.

4.2.2 Accuracy

Concerning the accuracy of the body detection SSD model under deployment, we have carried out a reference test on COCO 2017 evaluation dataset filtered on people. The resulted AP at $IoU\ threshold = 0.3$ is 0.198. Comparing with the latest people detection models trained by Xperience AI based on the backbone of yolox-nano, the body detection SSD model is much less accurate with smaller AP values. However, we still consider the detection accuracy as acceptable and take the body detection SSD model in the first implementation of the game system to demonstrate the feasibility of integration of the game algorithms.

5 Conclusion and Perspectives

In this internship, we have finished the development, prototype and testing of the end-to-end demo application "1-2-3 GAP" on GAP9_EVK. To begin with, we have designed the game system based on GAP processor, using requirement diagram, use case diagram, sequence diagram and finite state machine for each component. According to the system, we have reused and extended existing algorithms and codes - the neural network is taken from the existing body detection application on GAP8 (the previous generation of GAP), which was trained and quantized by Greenwaves' AI team. In the next step, we have done an evaluation on the body detection SSD model which is under deployment on GAP processors and obtained its AP=0.198 at $IoU\ threshold = 0.3$ from the precision-recall curve. An investigation on YOLOX models under training and evaluation by Xperience AI is also carried out during this internship, in order to find prospective new models to improve people detection accuracy on GAP9. Besides, we have also conducted an on-board test on reference speed for different NN models, i.e. body detection SSD model and YOLOX family. Based on the body detection and movement detection algorithms, we have carried out the software and hardware integration as well as system implementation of the game in C programming on GAP9_EVK with ext-devices, including camera, display, audio amplifier, servo motor and red laser pointer. In the end, we have tested the processing speed of different components of the game system and proved that the system is suitable for real-time deployment on GAP9_EVK at 5fps.

Potential extensions proposed on software and hardware perspect for the demo are listed below:

- Port the ILI9341 TFT display driver from GAPUINO8 to GAP9_EVK via SPI bus on MikroE socket.
- Connect to Audio Add-on board and play game sound via AK4332 amplified in PDM mode.
- Add the servo motor SG90 with a red laser pointer fixed on the output shaft to simulate the gun to eliminate players.
- Use GPIO pins to generate low-frequency (50Hz) PWM signal to control the servo motor.
- Design 3D model of the shelf to fix the relative positions of hardwares to ensure stable deployment.

In conclusion, in this internship we have made an integration of neural network referencing, sound and image processing on the edge device GAP9 to realise the functionality of the game system "1-2-3 GAP" and have achieved 5fps processing speed in real-time deployment, with limited

memory size (L1=128KB, L2=1.5Mb). The degree of novelty lies in the combination of body detection and frame difference algorithms to detect player's movement within each bounding box. Besides, as far as we know, the squid game implementation on a SoC processor expanded with a combination of ext-devices to achieve such a real-time interaction with players is also first in the market.

The perspectives of this internship project and work to be done in the future are listed as follows:

- The current body detection SSD model under deployment is not very accurate with low AP, and the problem of occlude and players coming across each other pose vulnerability to the detection. To improve accuracy of the people detection algorithm, we will need to test and verify the feasibility of using YOLOX models. Future work includes quantizing yolox-nano models and generating application code through GAPFlow to test on GVSOC platform in order to further verify the feasibility of deploying these models on GAP9, concerning inference speed and accuracy.
- People tracking algorithms can be added into the game system to identify player id and rule out the players in software such that the "dead" players are not going to be taken into account in the next frames.
- The application code compiler should be transformed to CMake in the future, which is an on-going task of Greenwaves' SDK team.
- It is possible to shoot marketing video of the game playing for GAP business promotion.

6 References

[1] Padilla, R., et al., "A survey on performance metrics for object-detection algorithms." In 2020 international conference on systems, signals and image processing (IWSSIP), pp. 237-242. IEEE, 2020.

[2] Redmon, J., Santosh, D., et al., "You only look once: Unified, real-time object detection." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788. 2016.

[3] Tsung-Yi L., et al., "Microsoft COCO: Common Objects in Context.", CoRR, 2014. Available at: http://arxiv.org/abs/1405.0312.

[4] Mykhaylo A., et al., "2D Human Pose Estimation: New Benchmark and State of the Art Analysis", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[5] S. Mittal, "Object Tracking Using Adaptive Frame Differencing and Dynamic Template Matching Method." PhD diss., 2013.

[6] Luxoparts, "SG90 Micro Servo", datasheet, 2017

[7] AsahiKASEI, "AK4332 Low-Power Advanced 32-bit Mono DAC with HP", 019003761-E-00 datasheet, 2019

Appendices

Appendix 1: Digital I/Os availability on expansion connectors

Ball Name	I/O Function				Availability on Headers / PTH		Availability on FFC		Availability on MikroBus™	Notes
	Alternate 0	Alternate 1	Alternate 2	Alternate 3						
D4	RESETN	-	-	-		CN9 Pin 4				GAP9 hardware reset (active low)
B2	SPI1_SCK	GPIO33	USART3_CLK	-	CN2 Pin 5				(L) Pin4(*) – Output only	(*)= Level-shifted to V_MikroE (3.3V or 1.8V)
E4	SPI1_CS0	GPIO34	-	-	CN2 Pin 4				(L) Pin 3(*) – Output only	(*)= Level-shifted to V_MikroE
A1	SPI1_SDO	GPIO38	-	-	CN2 Pin 1				(L) Pin 6(*) – Output only	(*)= Level-shifted to V_MikroE
E3	SPI1_SDI	GPIO39	-	-		~~CN3 Pin 5~~			(L) Pin 5(*) – Input only	(*)=Level-shifted from V_MikroE
B1	I2C0_SDA	GPIO40	-	-		CN3 Pin 4			(R) Pin 6(*)	(*)=L'shifted, but usable only if V_MikroE=3.3V
D2	I2C0_SCL	GPIO41	-	-		CN3 Pin 8			(R) Pin 5(*)	(*)=L'shifted, but usable only if V_MikroE=3.3V
F4	I2C1_SDA	GPIO42	-	-	CN4 Pin 10	CN5 Pin 9				For CN5 : Beware – this is seen from bottom side
C1	I2C1_SCL	GPIO43	-	-	CN4 Pin 8	CN5 Pin 7				For CN5 : Beware – this is seen from bottom side
K4	I2C2_SDA	GPIO44	-	-					(R) Pin 3(*) – Input only	(*)=Level-shifted from/to V_MikroE, remap...
G5	I2C2_SCL	GPIO45	-	-					(R) Pin 4(*) – Output only	...I/O to use as mikroBus UART_RX/TX
H5	I3C_SDA	GPIO46	I2C3_SDA	SPI0_SDIO2	CN10 Pin 6					If CSI-2 used, shared I2C bus
J5	I3C_SCL	GPIO47	I2C3_SCL	SPI0_SDIO3	CN10 Pin 5					If CSI-2 used, shared I2C bus
G6	SAI0_SCK	GPIO48	USART2_CLK	-	CN4 Pin 5					
G7	SAI0_WS	GPIO49	-	-	CN4 Pin 7					
K5	SAI0_SDI	GPIO50	-	-	CN4 Pin 6					
H6	SAI0_SDO	GPIO51	-	-	CN4 Pin 2					
J6	SAI1_SCK	GPIO52	-	-		CN5 Pin 6	FFC0 Pin 6	FFC1 Pin 10		For CN5 : Beware – this is seen from bottom side
H7	SAI1_WS	GPIO53	SPI2_CS1	(*) -		CN5 Pin 10		FFC1 Pin 7		For CN5 : Beware – this is seen from bottom side
K6	SAI1_SDI	GPIO54	SPI2_CS2	(*) SPI2_SDIO2		CN5 Pin 5	FFC0 Pin 4	FFC1 Pin 12		For CN5 : Beware – this is seen from bottom side
K7	SAI1_SDO	GPIO55	SPI2_CS3	(*) SPI2_SDIO3		CN5 Pin 1		FFC1 Pin 9		For CN5 : Beware – this is seen from bottom side
H1	SAI2_SCK	GPIO56	SPI2_SCK	-	CN6 Pin 3			FFC1 Pin 1		
H2	SAI2_WS	GPIO57	SPI2_CS0	(*) -	CN6 Pin 7			FFC1 Pin 3		
H3	SAI2_SDI	GPIO58	SPI2_SDI	-	CN6 Pin 6			FFC1 Pin 4		
H4	SAI2_SDO	GPIO59	SPI2_SDO	-	CN6 Pin 2			FFC1 Pin 6		
J4	USART0_RX	GPIO60	-	-	CN7 Pin 1					
J3	USART0_TX	GPIO61	-	-	CN7 Pin 4					
J1	USART0_CTS	GPIO62	-	-	CN7 Pin 2					
J2	USART0_RTS	GPIO63	-	-	CN7 Pin 3					
K1	FAST_REF_CK/U	GPIO64	-	-	CN0 Pin 2		FFC0 Pin 2			On EVK V1.0, also used as 3V3_CAM Enable
E5	PWM0	GPIO67	-	-	CN2 Pin 6				(R) Pin 1 – Output only	Usable as external reference clock to GAP9
D3	PWM1	GPIO68	-	-		CN5 Pin 4				Level-shifted to V_MikroE [CN2.8 on EVK v1.0]
G4	JTAG_TCK	GPIO81	USART4_CLK	-		CN1 Pin 8				For CN5 : Beware – this is seen from bottom side
E2	JTAG_TDI	GPIO82	USART4_RX	-		CN1 Pin 6				
F3	JTAG_TDO	GPIO83	USART4_TX	-		CN1 Pin 2				
D1	JTAG_TMS	GPIO84	USART4_CTS	-		CN1 Pin 3				
F2	JTAG_NTRST	GPIO85	USART4_RTS	-		CN1 Pin 7				
D6	WUP_SPI2_SCK	GPIO86/BOOT0	-	-					(L) Pin 2(*) – Output only	(*)= Level-shifted to V_MikroE (3.3V or 1.8V)
D5	WUP_SPI2_SDI	GPIO87/BOOT1	-	-		CN3 Pin 7			(R) Pin 2(*) – Input only	(*)= Level-shifted to V_MikroE (3.3V or 1.8V)
A2	WUP_SPI2_CS0	GPIO89	-	-	CN4 Pin 1	CN5 Pin 2				For CN5 : Beware – this is seen from bottom side

Appendix 2: Application code of the main function

```
static int start(){
    // Frequency settings
    pi_freq_set(PI_FREQ_DOMAIN_FC,FREQ_FC*1000*1000); // 400MHz
    pi_freq_set(PI_FREQ_DOMAIN_CL,FREQ_CL*1000*1000); // 400MHz
    pi_freq_set(PI_FREQ_DOMAIN_PERIPH, 400 * 1000 * 1000); // 400MHz
    // Memory allocation and hardware configuration
    char ImageName[30];
    uint8_t *ImageInChar[2];
    #ifdef SEND_TO_LAPTOP
    uint32_t bitstream_size = 0;
    uint8_t *jpeg_image = (uint8_t *)pi_l2_malloc(JPEG_BUFFER_SIZE);
    if (jpeg_image == NULL){
        printf("Failed to allocate Memory for jpeg_image buffer (%d bytes)\n",
               JPEG_BUFFER_SIZE);
        pmsis_exit(-6);
    }
    #endif
```

```
// Initiate I2S AK4332 amplifier
#ifdef GAME_SOUND
float speed = 1.0 ;
PRINTF("Initiating I2S...\n");
if (ak4332_init()){
    printf("Failed to open ak4332!\n");
    pmsis_exit(-6);
}
#endif
#ifdef FROM_CAMERA
// allocate 2 buffers of 640*400 for frame differencing
ImageInChar[0] = (uint8_t *)pi_l2_malloc(Wcam * Hcam * sizeof(uint8_t));
ImageInChar[1] = (uint8_t *) pi_l2_malloc( Wcam * Hcam * sizeof(uint8_t));
if (ImageInChar[0] == NULL || ImageInChar[1] == NULL){
    printf("Failed to allocate Memory for Image (%d bytes)\n",
           Wcam * Hcam * sizeof(uint8_t));
    pmsis_exit(-6);
}
// Calculate scale and offset for image resize
uint8_t scale = MIN((int)(Wcam/W),(int)(Hcam/H));
uint16_t Xoffset = (Wcam - W * scale)/2;
uint16_t Yoffset = (Hcam - H * scale)/2;
// Open camera
PRINTF("Opening CSI2 camera\n");
if (open_camera_csi2()){
    printf("Failed to open camera!\n");
    pmsis_exit(-1);
}
#else // Reading image sequence from host pc
...
#endif // <- FROM_CAMERA
// Initiation and memory allocation of CNN and SSD graphs
if(initSSD()){
    printf("SSD Init exited with an error\n");
    pmsis_exit(-6);
}
if (body_detectionCNN_Construct()){
    printf("CNN constructor exited with an error\n");
    pmsis_exit(-4);
}
// Game start
printf("\n========================= READY??! GO!!! =========================\n");
unsigned long long ti_start = pi_time_get_us();
while ((int)(pi_time_get_us()-ti_start) < GAME_TIME * 1000000){
    // Moving phase start (with game sound)
    #ifdef SERVO_CONTROL // turn off red laser pointer as default
    pi_gpio_pin_write(&gpio_lazer, PI_PAD_086, 0);
    #endif
    #ifdef SEND_TO_SCREEN
    writeFillRect(&ili, 0, 0, W_ili, H_ili, 0x0000);
    setTextColor(&ili, 0xFFFF);
    setCursor(&ili, 0, 0);
    writeText(&ili, "\n\n  RUN!!! \n", 6);
```

```
#endif
#ifdef GAME_SOUND
ak4332_play(speed);
if (speed<10.0) speed += 0.1;
#endif
// Still phase start
ti_awake = pi_time_get_us();
while((int)(pi_time_get_us()-ti_awake) < period * 1000000){
    /* Capture/read image and resize here, do frame differencing and find
     * moving players, draw bounding boxes on the previous frame and send to
     * display/laptop */
    buff_id = (img_id+1) % 2;  // use two buffers to capture image in turn
    #ifndef FROM_CAMERA  // read the next image from laptop
    ...
    #else // capture image from camera
    ov9281_capture(ImageInChar[buff_id]);
    // Crop and resize image captured from camera to 160x120
    for(int y=0;y<H;y++){
        for(int x=0;x<W;x++){
            ImageInChar[buff_id][y*W+x] =
             ImageInChar[buff_id][((y*scale+Yoffset)*Wcam)+(x*scale+Xoffset)];
        }
    }
    #endif // <- if FROM_CAMERA

    // Compare with previous frame, draw bounding boxes & send to screen/laptop
    if (img_id > 1){
        // Run frame differencing
        runFD(ImageInChar[1-buff_id], ImageInChar[buff_id], &bbxs);
        #ifdef SERVO_CONTROL // shoot red laser pointer
        shootLaserPointer(&bbxs);
        #endif
        // Print bounding boxes with status (survive/dead) of the previous frame
        printBboxes(&bbxs);
        printBboxes_forPython(&bbxs);
        // Draw BBs of the previous frame on the image
        drawBboxes(&bbxs, ImageInChar[1-buff_id]);
        #ifdef SEND_TO_LAPTOP
        /* Encode to jpeg and send to laptop, flush the image to the workstation
         * using semi-hosting here */
        ...
        #endif
        // Send to Screen
        #ifdef SEND_TO_SCREEN
        // resize the buffer for Adafruit TFT display of size 320*240 here
        ...
        pi_buffer_set_data(&buffer, ImageInChar[1-buff_id]); // set buffer data
        pi_display_write(&ili, &buffer, 0, 0, W_ili, H_ili);
        #endif
    }
    // Run CNN model
    pi_cluster_task(task, (void (*)(void *))RunNN, NULL);
    pi_cluster_task_stacks(task, NULL, CLUSTER_SLAVE_STACK_SIZE);
```

```
            pi_cluster_send_task_to_cl_async(&cluster_dev, task,
                                             pi_task_block(&wait_task));
            pi_task_wait_on(&wait_task);  // wait for the finish of RunNN()
            // Run SSD model for post-processing
            pi_cluster_task(task, (void (*)(void *))RunSSD, NULL);
            pi_cluster_task_stacks(task, NULL, CLUSTER_SLAVE_STACK_SIZE);
            pi_cluster_send_task_to_cl_async(&cluster_dev, task,
                                             pi_task_block(&wait_task));
            pi_task_wait_on(&wait_task); // wait for the finish of RunSSD()
            img_id++;
        }
    }
    printf("\n========================== GAME OVER!!! ==========================\t");
    // free CNN and SSD memory, close the cluster, print “GAME OVER!!!” on display
    ...
    pi_cluster_close(&cluster_dev);
    ...
    return 0;
}
```

Appendix 3: Power-Up Sequence of DAC and AK4332 Amplifier in PDM Mode

The PDN (power-down) pin of the AK4332 is connected on the Audio Add-On board, driven by GPIO1 of the GPIO expander IC implemented on the Audio Add-On board (IC9/FXL6408UMX). The AK4332 must be enabled by setting to Logic-1 its input pin PDN before being programmed and used; and disabled after use. Fig. 31 demonstrates the configuration sequence required to properly power up the different parts of the AK4332, as described below:

(1) After all power supplies are On, PDMCLK and DSDCLK should be input before powering up PLL or CP1.
(2) Set the PDN pin Logic-1 to release the power-down.
(3) Set DAC initial settings. (Write 0x02 data into address 0x26 and write 0xC0 data into address 0x27)
(4) Set sampling frequency (FS[4:0] bits) and the input signal path of the DAC (CM and FS).
(5) Set PDM bit to PDM 1-bit Mode or DSD Mode by PDMMODE bit.
(6) Set headphone amplifier volume by HPG[2:0] bits.
(7) Start the internal master counter by PMTIM bit.
(8) Power-up charge pump CP1 by PMCP1 bit and wait 6.5ms for CP1 output voltage stabilization.
(9) Power-up LDO1P and LDO1N by PMLDO1P bit and PMLDO1N, respectively) and wait 0.5ms for LDO output voltage stabilization.
(10) Power-up DAC by PMDA bit.
(11) Power-up p charge pump CP2 and wait 4.5ms for CP2 output voltage stabilization.
(12) Power-up headphone amplifier by PMHP bit. The power-up time of the headphone amplifier is 23.9ms (@fs = 48kHz).

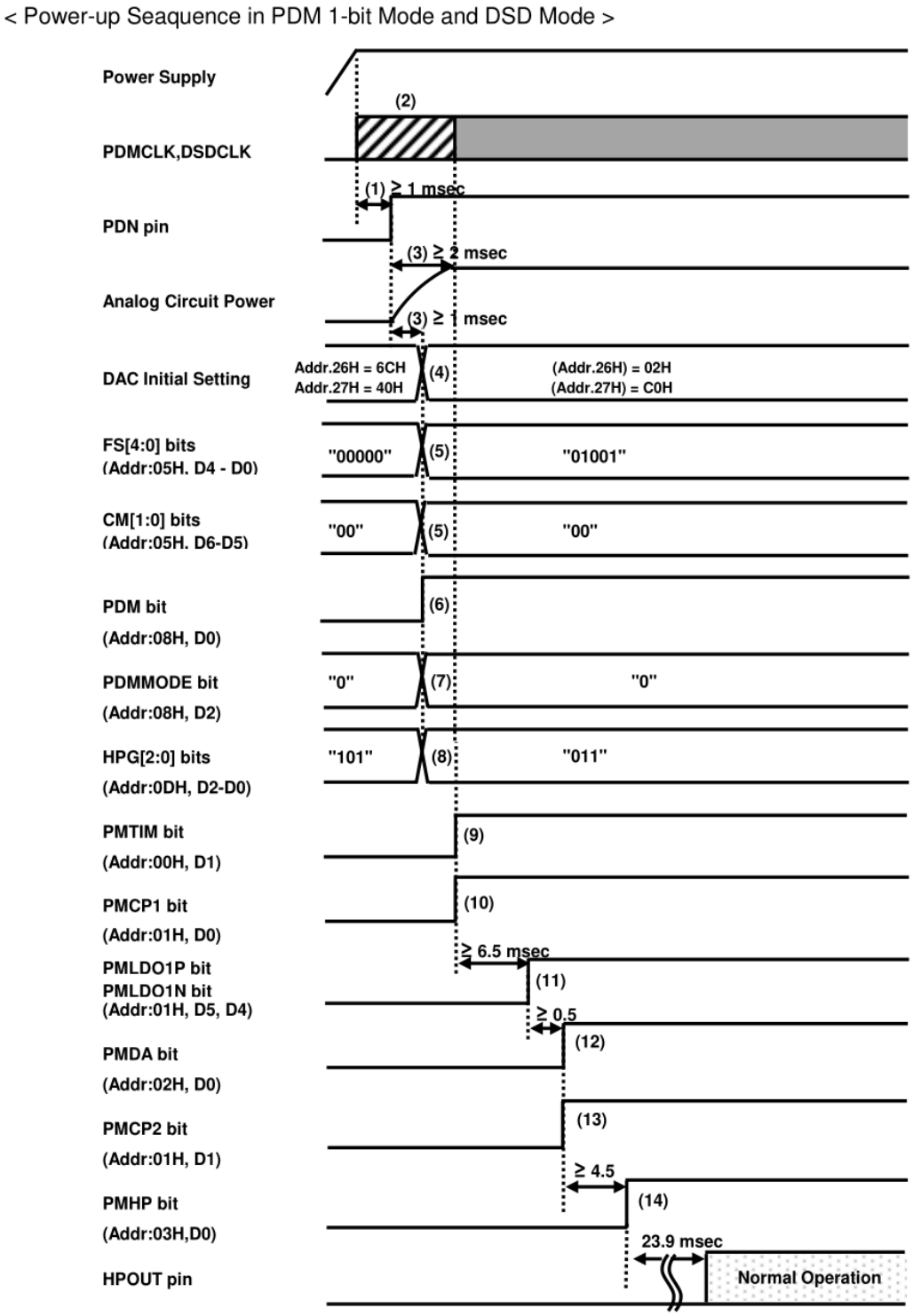


Figure 31: Power-Up Sequence Example of DAC and Headphone Amplifier in PCM 1-bit Mode and DSD Mode [7]

Appendix 4: Application code of servo task

```
typedef struct task_servo{
    pi_task_t mytask; // task start point
    pi_task_t mytask1; // pwm active
    pi_task_t mytask2; // pwm non-active
    pi_task_t mytask3; // red laser on
```

```
    pi_task_t mytask4; // red laser off
    pi_task_t mytask5; // update pwm active time
    pi_task_t mytask6; // signify end of servo task
    uint32_t minimum_last_time_us; // minimum lasting time of each PWM task
    uint16_t active_time_us; // active time of pwm signal
    uint16_t *x_pos; // list of x positions to shoot
    uint8_t queuing_task_num; // total number of x positions
    uint8_t
} task_servo_t;


void task_servo_init(task_servo_t *task, int *x_pos, uint8_t num, uint32_t
minimum_last_time_us){
    task->minimum_last_time_us = minimum_last_time_us;
    task->x_pos = x_pos;
    task->queuing_task_num = num;
    // starting point to launch the root task
    pi_task_callback(&(task->mytask), callback_function, (void *)task);
    // pwm active command
    pi_task_callback(&(task->mytask1), callback_function1, (void *)task);
    // pwm non-active command
    pi_task_callback(&(task->mytask2), callback_function2, (void *)task);
    // red laser pointer on
    pi_task_callback(&(task->mytask3), callback_function3, (void *)task);
    // red laser pointer off
    pi_task_callback(&(task->mytask4), callback_function4, (void *)task);
    // to update active time of pwm tasks
    pi_task_callback(&(task->mytask5), callback_function5, (void *)task);
    // to signal the completion of the whole servo task
    pi_task_callback(&(task->mytask6), callback_function6, (void *)NULL);
}

void callback_function1(void *arg){
    task_servo_t *task = (task_servo_t *) arg;
    // write Logic-1 to the PWM pin - pwm active
    pi_gpio_pin_write(&gpio_pwm, PI_PAD_067, 1);
    if (!task_servo_fini)
        pi_task_push_delayed_us(&(task->mytask2), task->active_time_us);
}

void callback_function2(void *arg){
    task_servo_t *task = (task_servo_t *) arg;
    // write Logic-0 to the PWM pin - pwm non-active
    pi_gpio_pin_write(&gpio_pwm, PI_PAD_067, 0);
    if (!task_servo_fini)
        pi_task_push_delayed_us(&(task->mytask1), 20000-(task->active_time_us));
}

void callback_function3(void *arg){
      task_servo_t *task = (task_servo_t *) arg;
      pi_gpio_pin_write(&gpio_lazer, PI_PAD_086, 1); // write Logic-1 to the RST pin
      // turn of red laser pointer after 2s
```

```
        pi_task_push_delayed_us(&(task->mytask4), 2000000);
}

void callback_function4(void *arg){
    task_servo_t *task = (task_servo_t *) arg;
    pi_gpio_pin_write(&gpio_lazer, PI_PAD_086, 0); // write Logic-0 to the RST pin
}

// update active time of pwm signal
void callback_function5(void *arg){
    task_servo_t *task = (task_servo_t *) arg;
    // convert x_pos to pwm active time
    task->active_time_us = 2100-6.25*(*(task->x_pos));
    (task->x_pos++); // move pointer to the next x_pos
}

// Signal the complete of the task
void callback_function6(void *arg){
    task_servo_fini = 1;
}

void callback_function(void *arg){
    task_servo_t *task = (task_servo_t *) arg;
    // Push the first task
    // update active time of pwm signal
    pi_task_push(&(task->mytask5));
    // Generate PWM signal, period=20ms, active time 1-2ms
    pi_task_push(&(task->mytask1));
    // turn on the red lazer pointer after 1s
    pi_task_push_delayed_us(&(task->mytask3), 1000000);

    // Arrange to push the queuing tasks in turn after waiting for a
       minimum_last_time
    int x_pos_id = 1;
    if ((task->queuing_task_num)>1){
        for (x_pos_id=1;x_pos_id<(task->queuing_task_num);x_pos_id++){
        // update active time of pwm signal
        pi_task_push_delayed_us(&(task->mytask5),
                                x_pos_id*(task->minimum_last_time_us));
        // turn on the red laser pointer after 1s
        pi_task_push_delayed_us(&(task->mytask3),
                                x_pos_id*(task->minimum_last_time_us)+1000000);
        }
    }
    // Signal complete of the servo_task
    pi_task_push_delayed_us(&(task->mytask6), x_pos_id*(task->minimum_last_time_us));
}
```