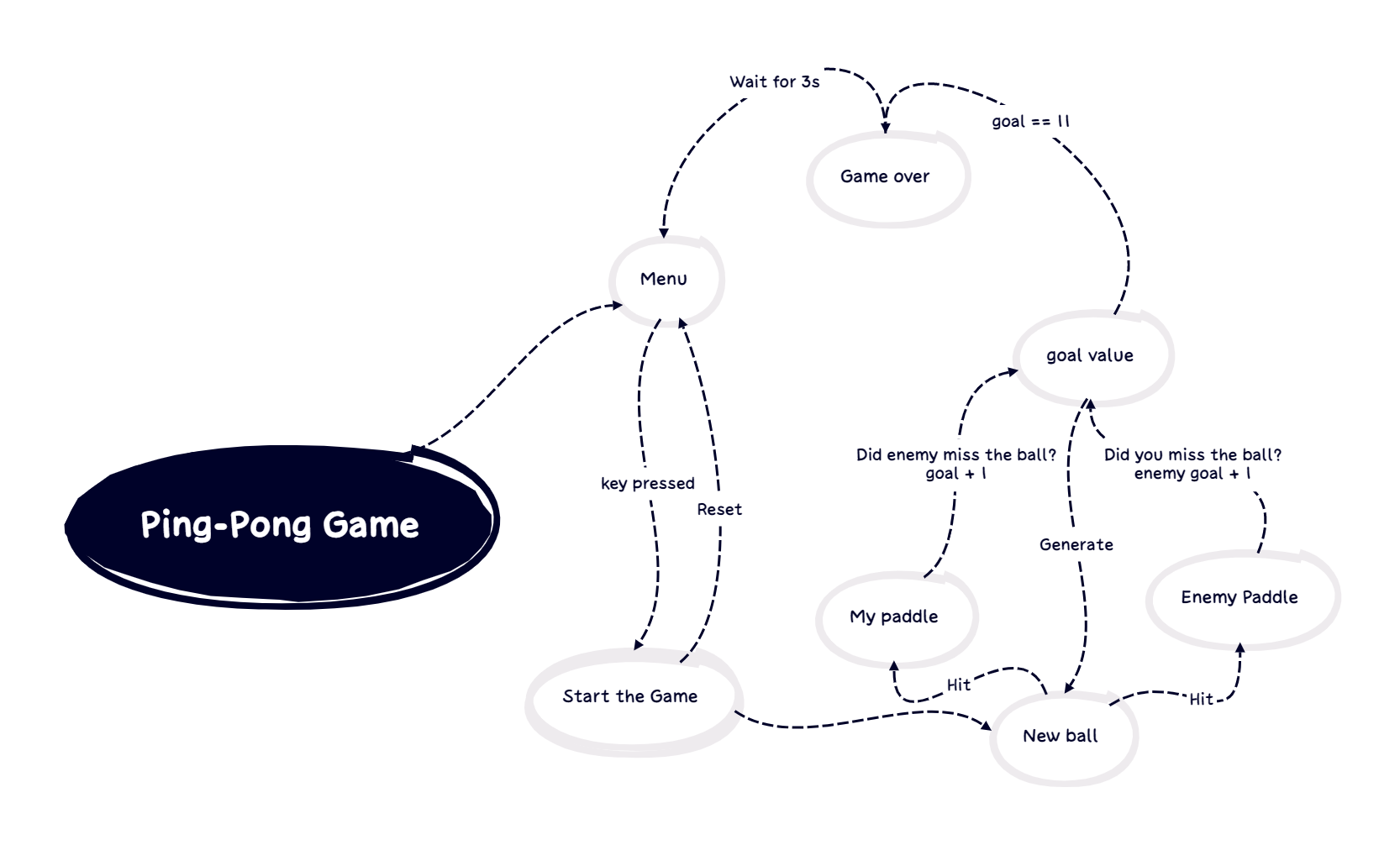
# ELEC5620M Mini-Project Repository

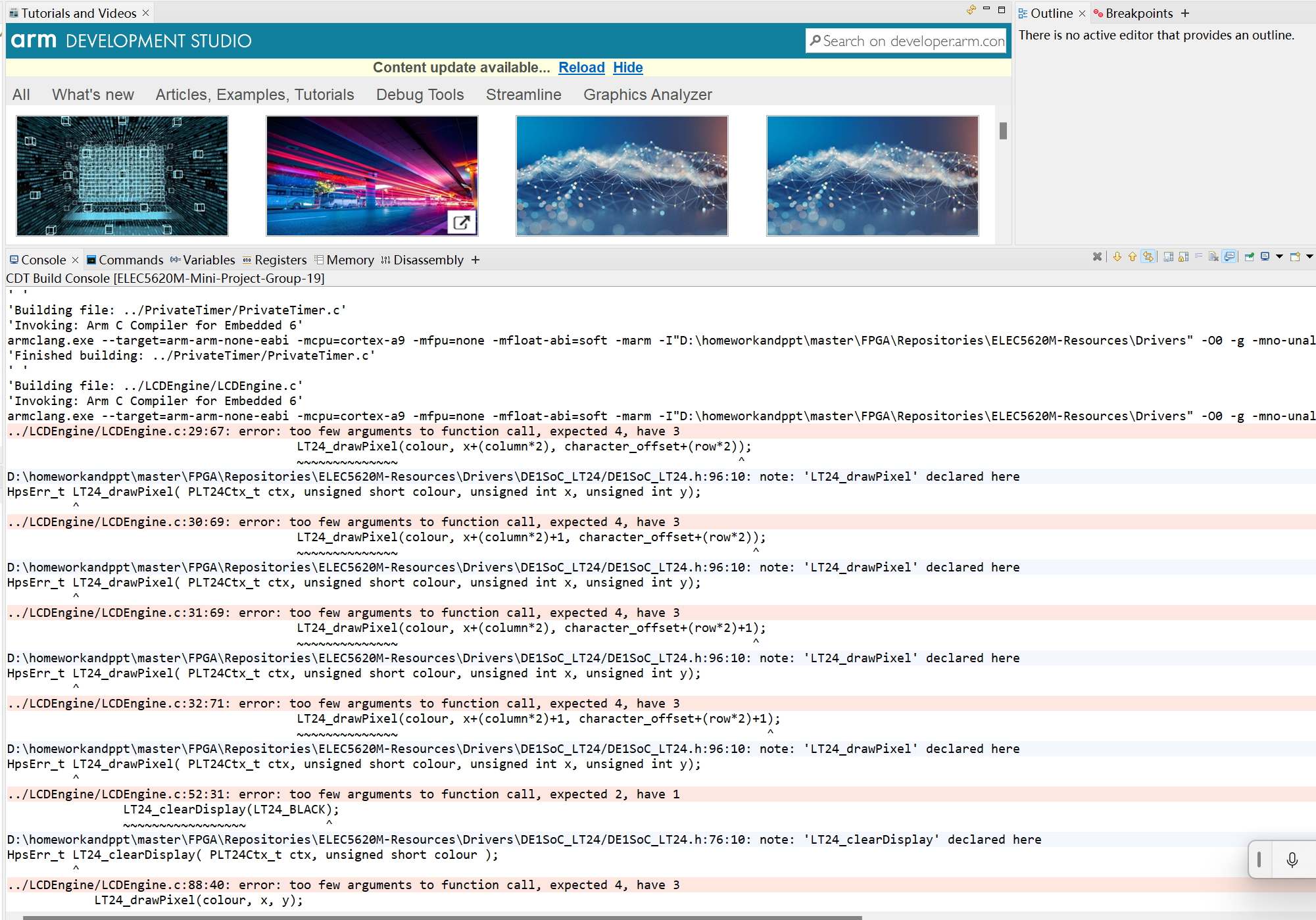
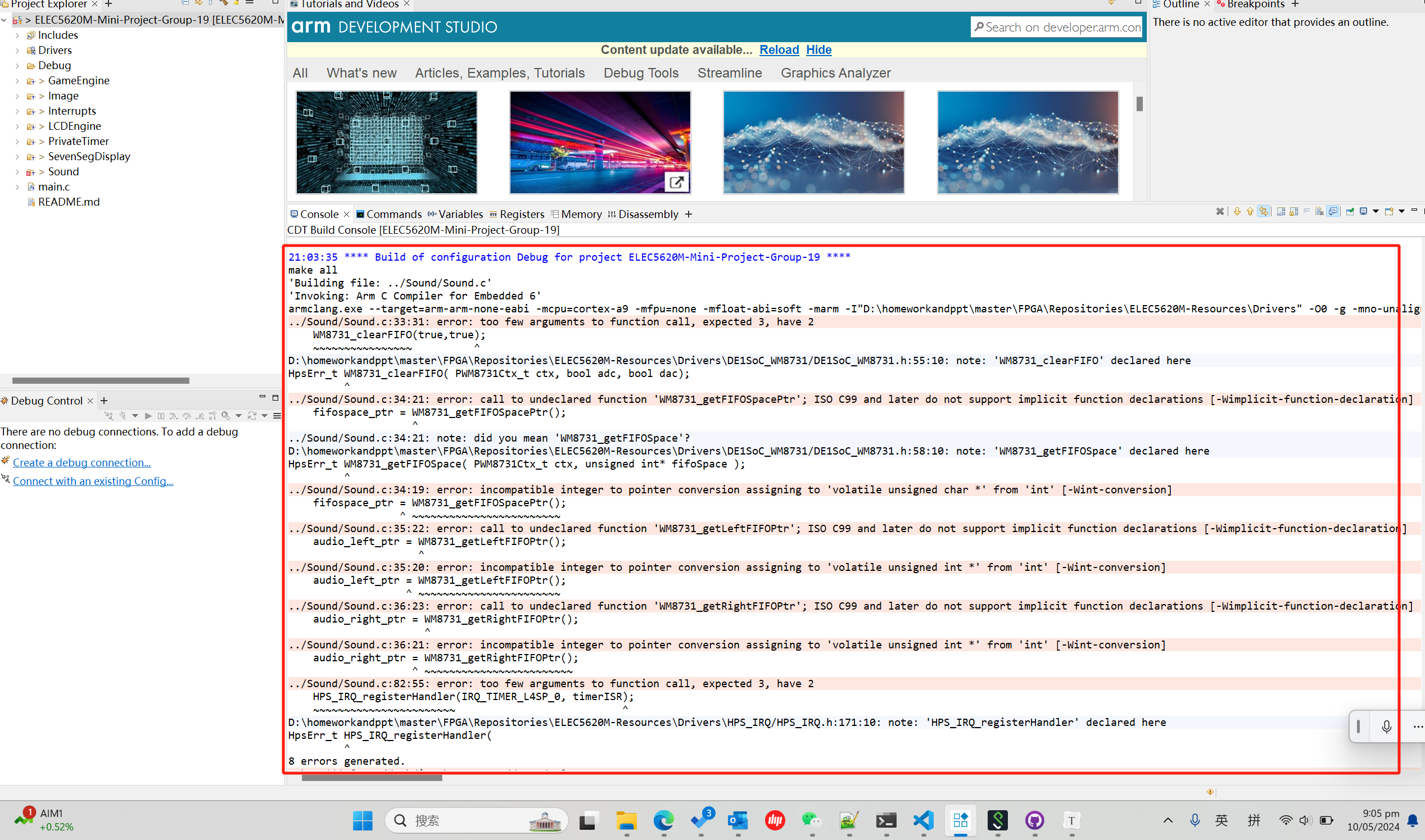
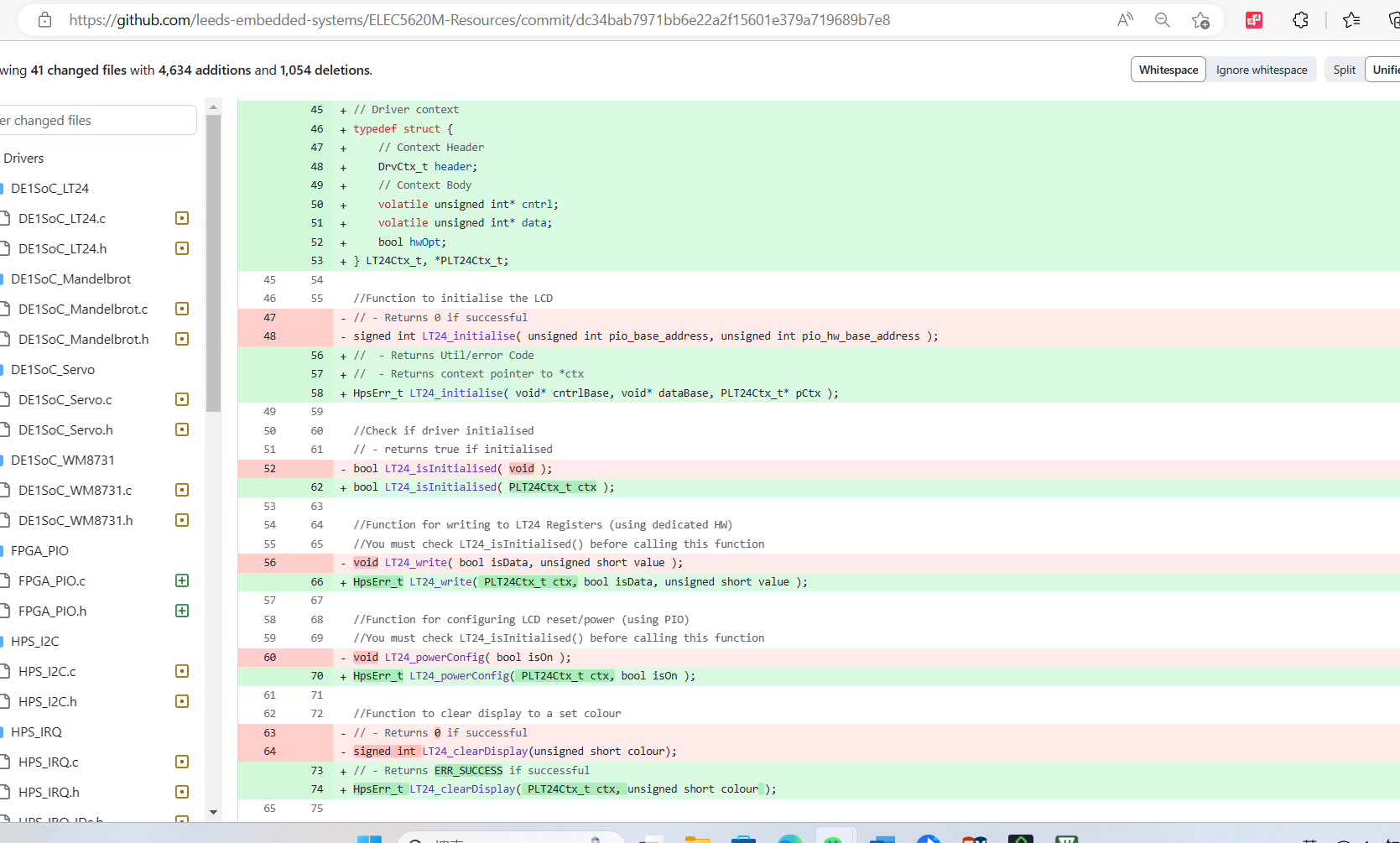
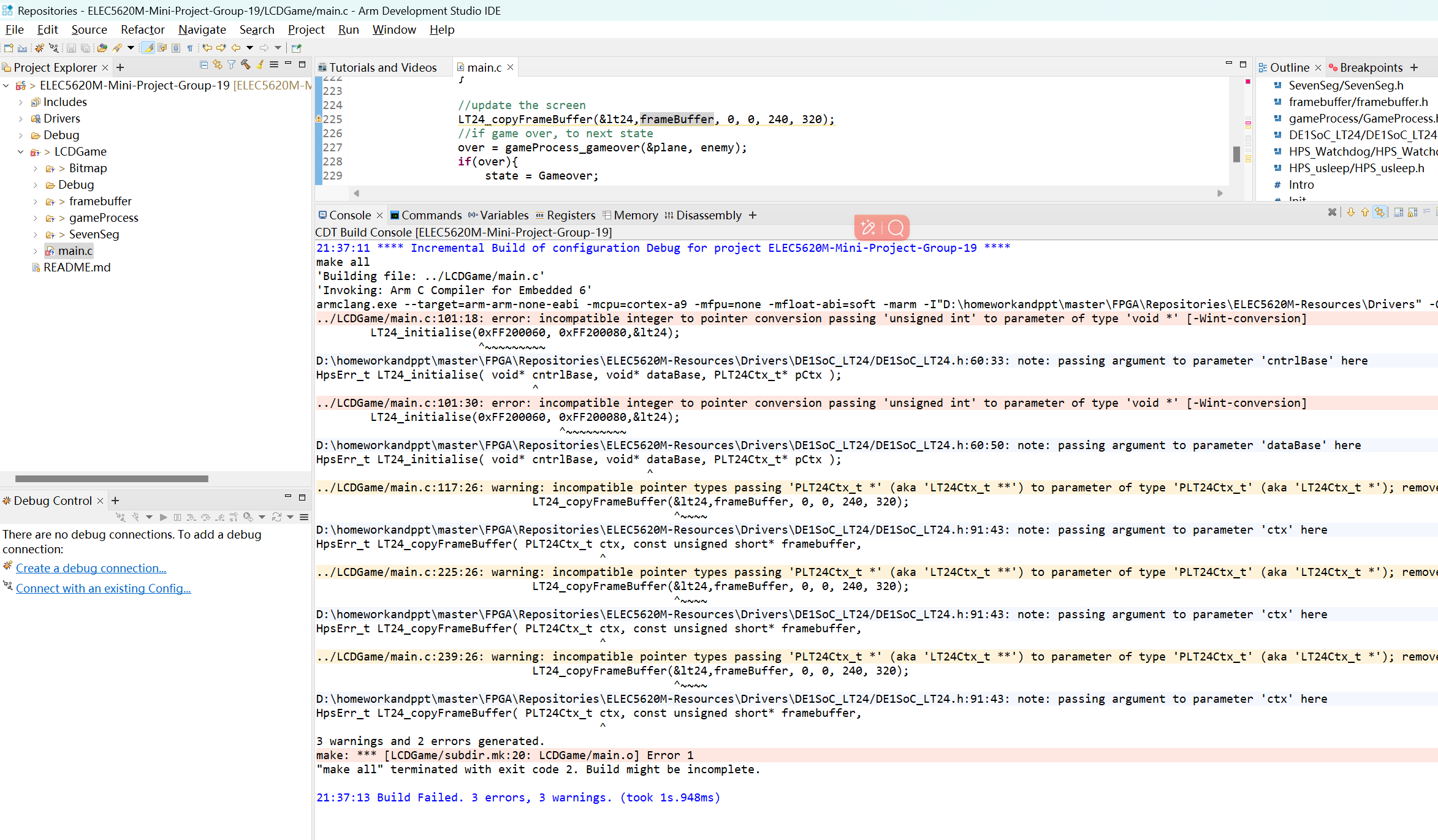
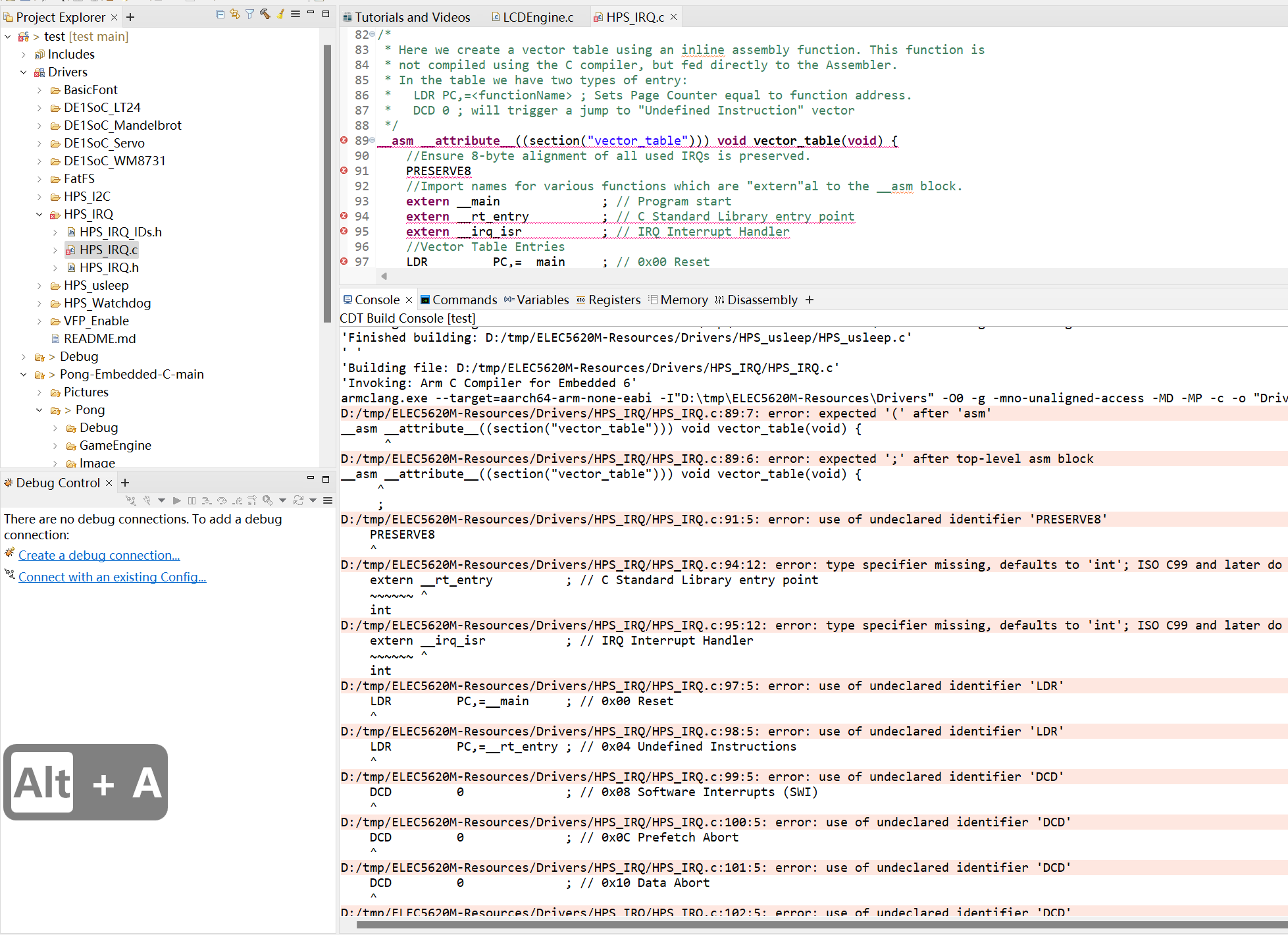
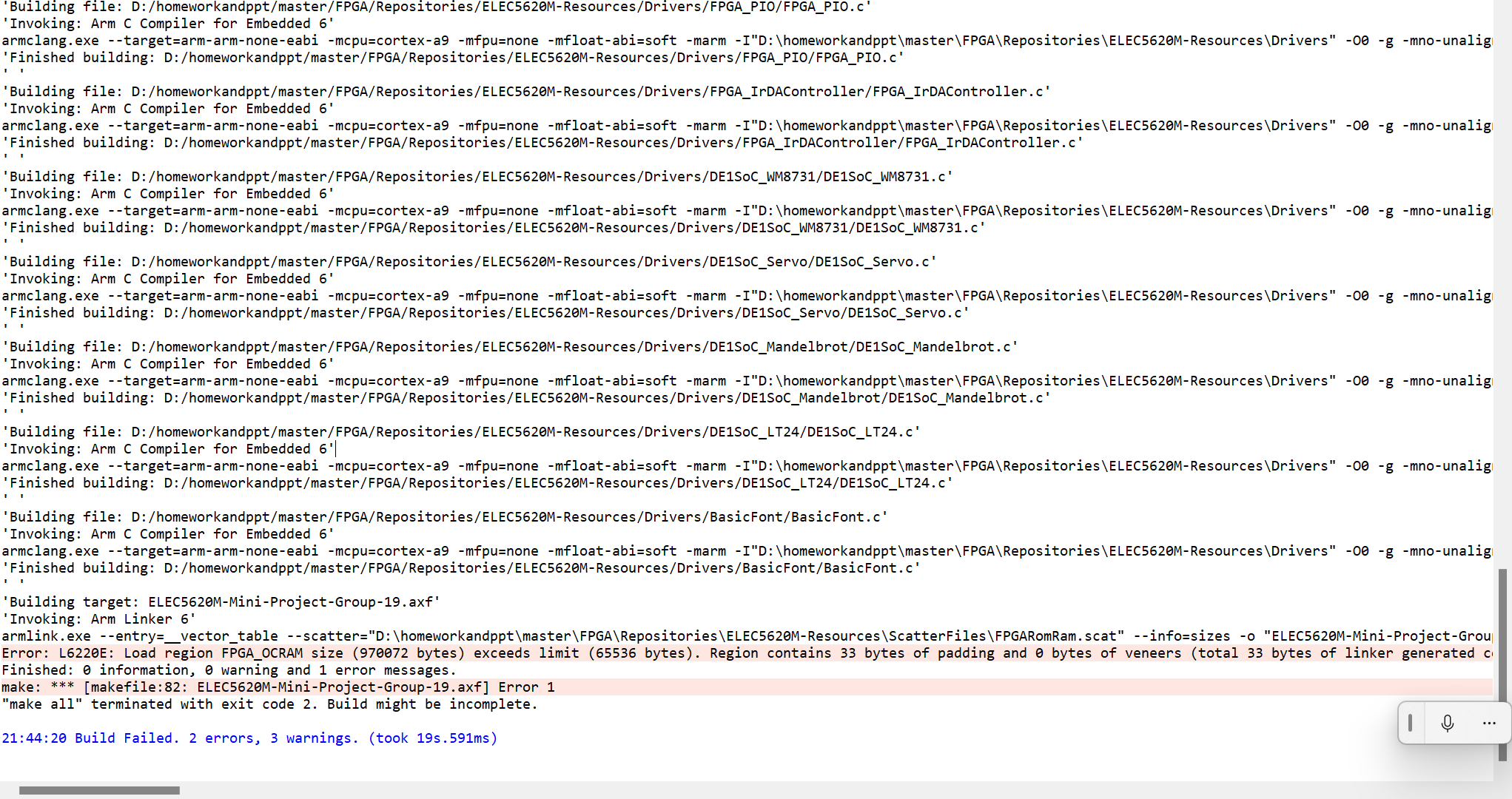
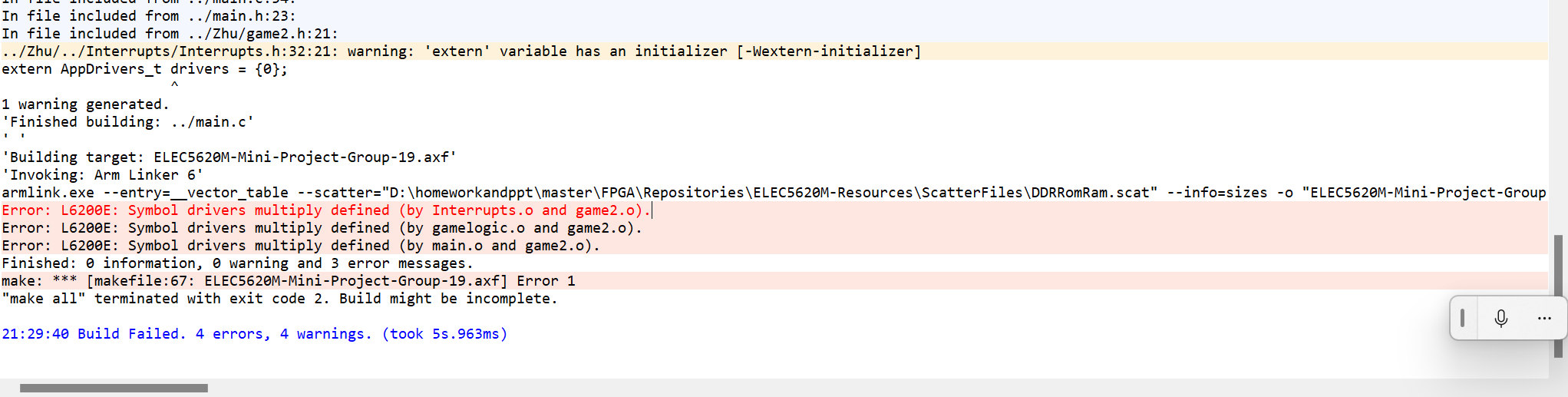
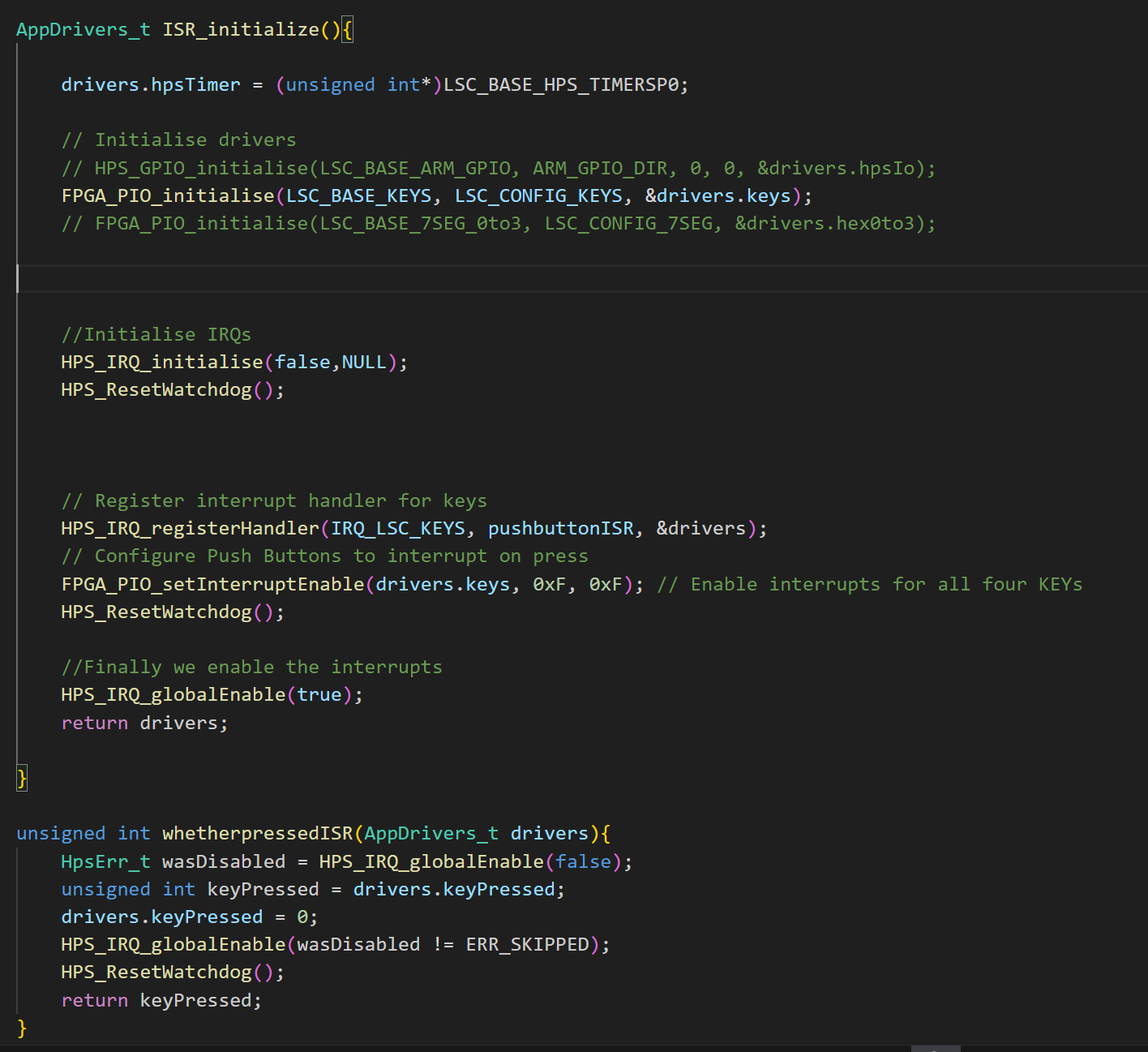
This is the blank repository in which you must incrementally commit your code.

#### PingPong Game （201715540）

* The state machine plan for the pingpong ball is shown below:



#### Debugging(201715540)

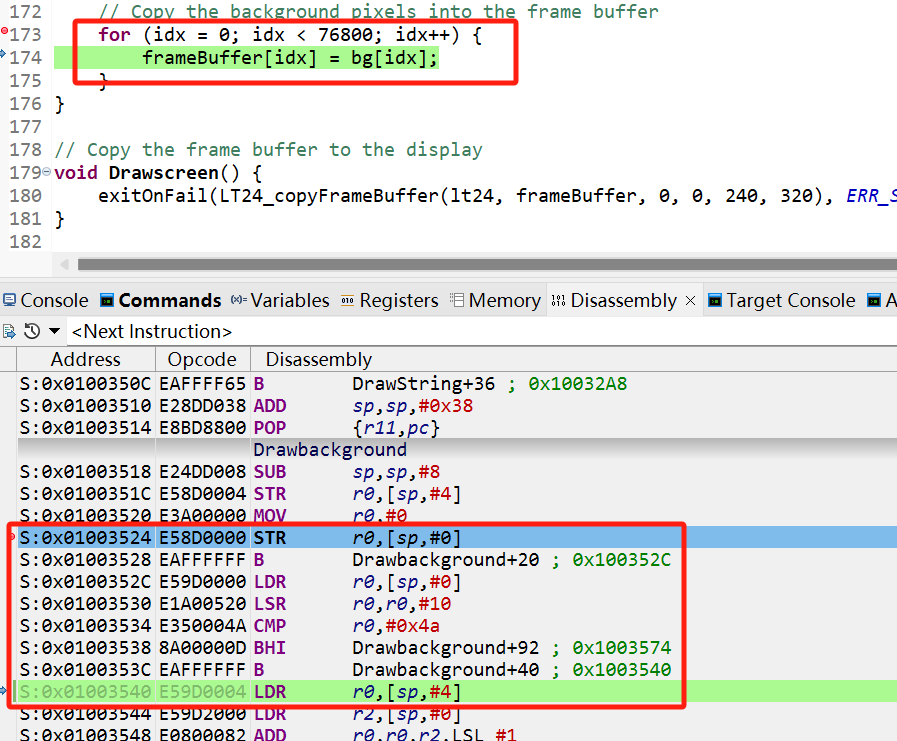
* I was mainly responsible for the integration and implementation of the pingpong game with the LCD and the whole project, and implemented the interrupt mechanism to make the buttons more responsive. I also encountered a lot of problems in the process:
  + Firstly, in the process of adapting the pingpong code to the LCD code, there were many times of ERROR due to the incorrect number of parameters in the LT24 related driver code caused by the driver code update (as shown in the figure below).
  + 
  + Then a similar problem occurred during the porting of the audio (shown below).
  + 
  + In the analysis and discussion together found in this year's update, for the LT24 update is mainly to add a context of the data structure LT24Ctx\_t (as follows), the purpose of this is to put the LT24 control state and data into a structure, so that the entire LT24 driver to achieve a more fluent and clear.
  + 
  + It was also discovered while porting that you cannot simply pass the address directly as a parameter, but rather you have to pass it with a pointer that represents the address so that you can accurately use the LT24's display driver.
  + In a later port, I discovered that the driver didn't just add a context, it also added an IRQ to represent interrupts and various event handlers, and to take full advantage of this aspect of the update, I tried to make the keys use interrupts.
  + 
  + When using the LT24 driver for drawing, when we create an array larger than 65536 bytes, it will trigger an error message causing the compilation to be unsuccessful, only to find out that we need to change the FPGARomRam.scat to DDRRomRam.scat, so that we can store the picture in the DDRRom instead of only 65536 bytes in the Rom.
  + 
  + When debugging the interrupt code and adapting it to the pingpong game, there is often a problem with the output of !EXCEPTION! In Abort Mode. Later on, after reviewing the information, I found that there was a wild pointer, which led to an error caused by passing a wild pointer into the function when initialising the LCD.
  + In the initialization of drivers encountered a lot of problems, a lot of files have defined drivers lead to multiple places to repeat the definition, this is the merging of the code encountered the most problems, that is, the abuse of global variables lead to drivers can not play a role in the correct, and finally through the main.h in the definition of the global variable drivers, and then elsewhere using the extern to call drivers to realize the shared drivers.
  + 
  + There is a problem that I have not been able to solve: that is, the key interrupt involves the initialization and registration of the event function can not be modularized as a whole written as a function to call, when the initialization of the cross-file call, the interrupt will fail, this is a very troubling problem, the following figure, when I defined the function in the interruption.c in the main call this function It doesn't work, only when I run these instructions under the current file, it seems like the interrupt will work.
  + 

#### Hardware Testing（201715540）

* After dealing with the problem of mismatched parameter numbers caused by the update of LT24's driver, we compiled and burned the code into the development board, we could see that we successfully entered the cover and entered the game, but the cover did not disappear with the start of the game and the pingpong ball kept blackening the background in the process of movement. Later, we learnt that the student in charge of the pingpong game had set the background to black by default, so that every time the paddle and the ball moved, the previous area would be blacked out, so we chose to change the background back to black. docx/testvideo1.mp4
* After fixing the background, we noticed that the movement of the ball also blacked out the text (as shown in the video below), so we chose to remove the text.Docx/testvideo2.mp4
* After dealing with the above problem, I found another problem with the scoreboard displaying abnormally, and after careful analysis, I found that it was the scoreboard changing without effectively clearing the previous number.Docx/testvideo3.mp4
* After solving the above problems, I found that I can finally play the game normally, the next step is to add the corresponding background image and adapt the game2.Docx/testvideo4.mp4
* After merging the game2 code and adding the new pingpong game cover, the LCD code was also updated, but since the LCD code puts all the displays in the framebuffer, it slows down the game, as shown in the following video: Docx/testvideo5.mp4
* After informing the relevant LCD students to modify the code, the problem was successfully solved, the game speed back to normal, basically debugging is complete, but the game button due to interruptions can not be recognised as a long time to press the state, which leads to the ping-pong paddle can only be moved one by one press. The following video shows: Docx/testvideo6.mp4
* After processing the interrupt key detection and ping pong paddle key detection separately, the continuous movement of the ping pong paddle was finally achieved and the game failure screen was successfully displayed as shown in the video below:

## Analyse

#### LT24 Driver Code Analysis (201715540)

* • LT24\_initData variable
* – This variable is used to initialise the LCD screen and configure the relevant registers of the LCD, and is mainly used in the LT24\_initialise function.
* • LT24Ctxt*, \*PLT24Ctxt\* type
* – Context type for LT24, containing header header variables, cntrl control variables and data data variables, then finally whether hwOpt is enabled for hardware variables.
* • LT24\_initialise function
* – The function has three parameters, which are analysed as follows:
* • cntrlBase: void\* type, meaning the base address of the controller, mainly used for the configuration of the LCD screen.
* • pCtx : PLT24Ctx\_t\* pointer type, this variable is mainly the context pointer of the LT24.
* • dataBase: void\* type, this type mainly represents the base address of the data, which is mainly used for LCD screen.
* – First the function uses the DriverContextAllocateWithCleanup function to reallocate the driver context for pCtx.
* – Then set a local context variable ctx and assign variables such as controller base address and data base address to ctx.
* – Let the *LT24CMDDATMASK bit, LT24LCDON bit,* LT24RESETn bit and LT24HWOPT(1) corresponding to the cntrl pointer under ctx be set to 1
* • *LT24CMDDATMASK: `(*LT24CSn | *LT24RDn |* LT24RS | LT24\_WRn | 0x0000FFFF)` indicates that the Write Enable bit, Read Enable, and Chip Select will be written, that is, positions 16-19 1.
* • LT24LCDON: Bit 21, indicates whether the backlight is backlit or not. To set 1.
* • LT24\_RESETn: bit 20, indicates whether to reset, set to 1.
* • LT24HWOPT(1): Hardware Optimisation, 1 Selection of Professional Hardware.
* – regVal &= ~(LT24\_CMDDATMASK | LT24\_LCD\_ON | LT24\_RESETn | LT24\_HW\_OPT(1));: Next, the bitwise inverse operation ~ is used to clear a specific bit in regVal to zero. The inverse bitwise operation ~, combined with the inverse bitwise operation &=, is used here to clear the bits corresponding to LT24*CMDDATMASK, LT24*LCD*ON, LT24*RESETn, and LT24*HW*OPT(1) to zero, which is equivalent to setting these bits to zero.
* – regVal |= (LT24\_CSn | LT24\_WRn | LT24\_RDn | LT24\_HW\_OPT(ctx->hwOpt));: The specific bit in regVal is then set to 1 by the bitwise-or operation. the bitwise-or assignment operator |= is used here to set the LT24*CSn, LT24*WRn, LT24*CSn, LT24*WRn, LT24*RDn, and the bit corresponding to LT24*HW\_OPT determined from the value of ctx->hwOpt are set to 1 using the bitwise or assignment operator |=.
* • LT24*CSn, LT24*WRn, LT24\_RDn are the chip select, write enable and read enable signal bits that control the LCD, and a setting of 1 means it is in an idle state.
* • LT24\_HW\_OPT(ctx->hwOpt) Determines whether the hardware option is enabled or not based on the value of ctx->hwOpt, and if it is enabled, sets the Hardware Option bit to one.
* – ctx->cntrl[LT24\_PIO\_DATA] = regVal;Finally, write the updated regVal back to the LCD controller's PIO data register to complete the initialisation setting of the register.
* – \_LT24\_powerConfig(ctx, true);soft reset the screen by calling this function to make the screen light up or go off.
* Code Optimization
  + In LCD related code, in order to realize framebuffer with many pictures pixel filling, we may involve many for loops and while loops, at this time we often because of framebuffer filling is too slow to cause the game running speed down, in order to improve the running speed, we need to streamline the code to achieve, so we for loop So we analyze the for loop as follows:
    - For example, in the following loop, in order to traverse the background pixels, we need to traverse 76800 times, and each time we need to run 8 assembly instructions.
* 
  + After we turn the for loop into a memcpy, we only need to execute 6 assembly instructions to accomplish what it takes 76,800 loops to do. That's a huge reduction in time (see below).
  + 