Debug Framework for SOC Based Windows Tablet and Developing Multimedia Apps

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Mitsu Shah

(14MECE23)

Electronics & Communication Engineering Branch

Electrical Engineering Department

Institute of Technology-Nirma University

Ahmedabad-382 481

May 2016

Debug Framework for SOC Based Windows Tablet and Developing Multimedia Apps

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Mitsu Shah
(14MECE23)

Under the guidance of

External Project Guide:	**Internal Project Guide:**
Neeraj Singh	**Dr. N. P. Gajjar**
Platform Architect,WSDS-CCG,	Assistant Professor, EC Department,
Intel Technology India Pvt. Ltd.,	Institute of Technology,
Bangalore.	Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch

Institute of Technology-Nirma University

Ahmedabad-382 481

May 2016

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Mitsu Shah

14MECE23

Disclaimer

Certificate

This is to certify that the Major Project entitled **"Debug Framework for SOC Based Windows Tablet and Developing Multimedia Apps"** submitted by **Mitsu Shah (14MECE23)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. N. P. Gajjar
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head, EC

Dr. P.N.Tekwani
HOD

Dr. P.N.Tekwani
Director, IT

Certificate

This is to certify that the Major Project entitled **"Debug Framework for SOC Based Windows Tablet and Developing Multimedia Apps"** submitted by **Mitsu Shah(14MECE23)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Neeraj Singh
Platform Architect,WSDS-CCG
Intel Technology India Pvt. Ltd.
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**-HOD & Director of NU-IT, **Dr. D.K.Kothari**-EC Section Head and **Dr. N.P.Gajjar**-PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for their guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. N. P. Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to **Kallapur, Madhusudan V**,Engineering Manager,Intel Technology India Pvt. Ltd. for his cordial support, constant supervision as well as for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages.I would also thank to **Singh, Neeraj K**,my Project Mentor for always helping,giving me good suggestions, solving my doubts and guide me to complete my project in better way.

I am obliged to **J, Vinithra** team member of System Integration and Debug team, Intel Technology India Pvt. Ltd. for the valuable information provided by her in respective fields. I am grateful for her cooperation during the period of my assignment.

Lastly, I thank almighty, my parents, brother and friends for their constant encouragement without which this assignment would not be possible.

- Mitsu Shah

14MECE23

Abstract

Debugging process has become a very challenging task as technology at platform and software level is increasing. Default debugging tools are not satisfying each and every requirement for upcoming new platforms. So existing tools need to be extend and need to develop new generic tools. So debuggers can debug new technology windows based platform very effectively in less time at kernel level.

In this project efforts are given to make debugging process smarter with enabling new exciting real sense features.

”Windbg” is a windows based debugger which is used to debug windows based platforms at both user and kernel level. I have developed new commands in DLL extension for windbg to extend this debugger and can satisfy all debugging requirements for upcoming new technology based on windows platforms. The ”Debug Utility” is a standalone tool which has all the commands those are extended for windbg. But this utility directly runs on the targeted system without needing any host or connection and eliminate all dependencies. The ”Energy Monitoring Tool” has been developed to monitor battery, charging, thermal and other battery related platform information continuously.

The multimedia apps developed in this project have enabled new exciting features of latest technologies and giving amazing experience to users. Bi camera capturing in a single window with single encoded file is capturing dual mode camera preview in a single window with picture in picture format and simultaneously saving it in single file. This app can be used by debuggers to debug new platforms where this new camera feature will be added. Emotion Detection app is detecting humans emotion in real life. Secure System app is securing the individuals system by recognizing unauthorized face.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

New technology is being developed day by day at both software and hardware level. As new technology comes, complexity is also increase at great extend. So debug process is becoming very challenging task for any developer. Because of that existing debugging tools need to be developed in such a manner that it can debug each devices individually at kernel level. With development in existing tools, we also need to develop new debug tools which includes information about new features in generic manner. Development of new multimedia apps are also necessary as new real sense and 3D camera technology are becoming more popular in market.

1.2 Objective

The main objective behind this project is to extend new features to introduce more smoother, finer and easier debug tools for SOC and to develop new features in multimedia framework which in turns used by both users and debuggers for enhancing new real sense technology.

1.3 Scope

The all developed debug utilities and DLL extension plugins can be used to debug new technology platforms with great extend and also be used as an alternative to other existing debug tools at Intel. The all newly developed multimedia Apps can be used by users to experience new exciting features and also be used by debuggers for checking their related test cases.

1.4 Requirements

Knowledge: Platform architecture, each device on SoC, Basic flow, config file, make file, multimedia frameworks
Languages: C, C++
HWs: Real Sense 3D camera, Intel Platforms

1.5 Complete project flow

The timeline of project work from the start of the project is shown in below gantt chart.

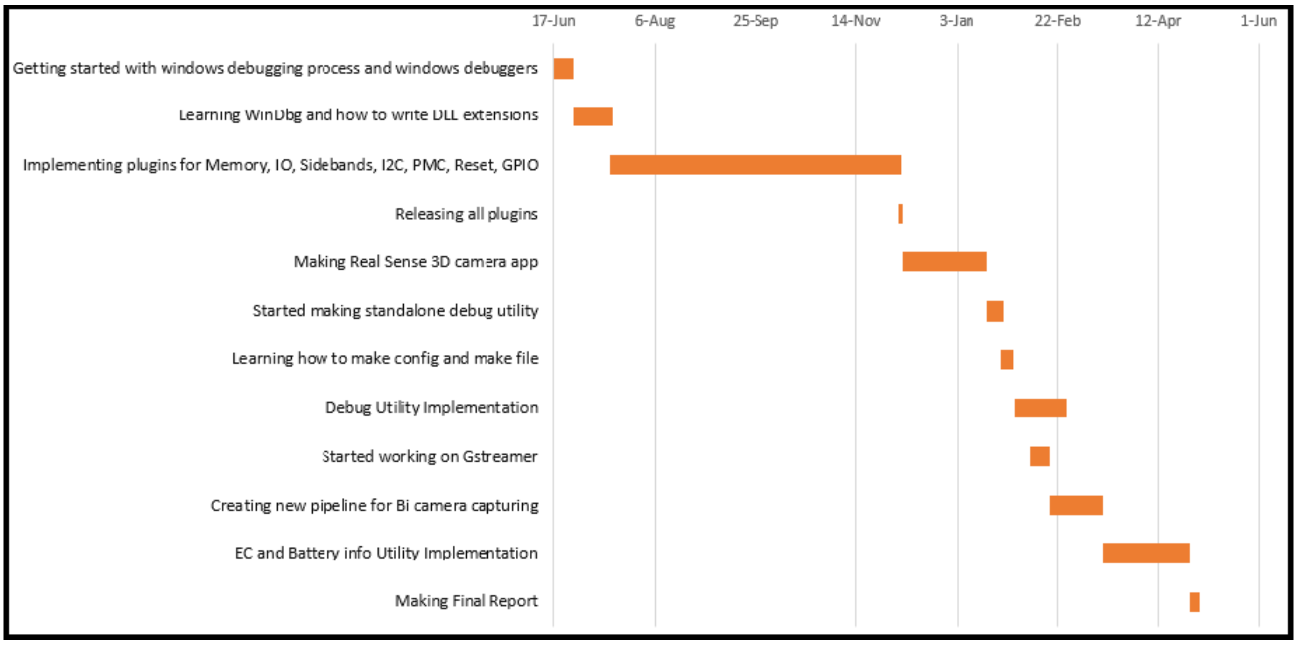


Figure 1.1: Gantt Chart

1.6 Thesis Organization

Chapter 2 describes the literature survey of windows debuggers, complete debugging process, types of debuggers, multimedia framework, realsense technology.

Chapter 3 describes the WinDbg which is one of the windows debugger. Its features, types of commands and what is DLL extension for windbg is described.

Chapter 4 describes the GStreamer framework, its features and multimedia technology supported.

Chapter 5 describes all newly implemented windbg DLL extensions which includes memory, IO, GPIO, I2C, Reset, sidebands, PMC and power unit related commands.

Chapter 6 describes standalone debug utility which is newly developed for debugging all devices.

Chapter 7 describes energy monitoring tool which is developed to debug battery, charging, thermal and EC related information.

Chapter 8 describes the multimedia app which is developed to capture dual camera at a same time, previewing on a single window in a picture in picture format and simultaneously save it in a single file.

Chapter 9 describes the newly developed realsense apps.

Chapter 10 conclusion is presented.

Chapter 11 future scope of this project is presented.

Chapter 2

Literature review

2.1 Windows debugging process

Debugging is a process of finding faults, bugs and cause of failure in system. Debugging can be at hardware level or at software level. Windows debugging process involves debugging of each and every components on windows based platforms, how their drivers are working in different and all.
To perform debugging process, debuggers are needed. Each company has its own developed debugger tool that is used for debugging. Windows also provides different debugging tools which can be used by developers.

2.2 Debugging tools for windows

There are different types of debugger provided by windows. They all are integrated into windows driver kit which provides the driver development environment. There are six different debugging environments that you can use after installing windows driver kit and visual studio. [1]

- Microsoft Windows Debugger (WinDbg)
- Microsoft kernel Debugger (KD)

- NTKD
- Microsoft Console Debugger (CDB)
- Microsoft NT symbolic Debugger (NTSD)
- Visual studio with integrated windows debugger

2.2.1 Microsoft Windows Debugger (WinDbg)

Windbg is a windows based debugger that can perform debugging process in two modes very powerfully. Those modes are user mode and kernel mode. It performs the debugging process for windows kernel, drivers, and system services along with user mode applications. Windbg uses symbol for source level debugging. All symbols are in the format of visual studio. There are many types of symbol files. From that only 2 types of symbol files it can access, those are PDB symbol files and public functions name which were complied with windows .dbg files. In windbg, we can write and edit source code, set breakpoints, view different memory section with memory leak and memory heap, variables and stack traces. Windbg has many commands. Windbg also provide the remote mode debugging for both kernel mode and user mode cases. windbg is based on graphical interface which is not provided in CDB/NTSD and KD/NTKD.[1]

2.2.2 Microsoft Kernel Debugger (KD)

KD is almost same as Windbg in functionality. Only difference is that KD is a character based console program while Windbg is based on graphical interface. KD is doing all kernel based analysis on all NT based OS. Multiprocessor debugging is possible on KD. You have to do debugging process on another system means you cannot do kernel debugging on the same computer where it is being debugged. You need two separate computer as KD does not support it.[1]

2.2.3 NTKD

NTKD is same in all manner to KD. Only one exception is that, when we open it, it spawns a new text window and on the other side, KD opens with command prompt window from which it is invoked.[1]

2.2.4 Microsoft Console Debugger (CDB)

Microsoft Console Debugger (CDB) is used to do low level analysis. It performs user mode debugging for memory and constructs. CDB is also character based console program like KD. CDB is very important for debugging a currently running program, recently crashed and live analysis. It is very easy to set up it as it is very simple. We can use it to check working application. When any application failed, at that time CDB can be used to do stack tracing and check at the bad parameters. By using CBD, one can perform many operation like display code, execute code, set breakpoints, examine memory values and also can change it. CDB is disassembling the code and displaying the assembly instructions. Source code can also be analyzed directly. It is possible to use memory locations through address and global symbols in CDB so one can refer the data and instructions by name in place of address. It is also possible to do debugging multi threads and process on CDB. CDB can access both paged and non-paged memory.[1]

2.2.5 Microsoft NT symbolic Debugger (NTSD)

NTSD is also same in all manner with CDB. Only one exception is that, when it is opened at that time it appears with new text window. On the other side, CDB opened in command prompt window from which it is being called.[1]

2.2.6 Visual studio with integrated windows debugger

If we download windows driver kit with windows debugger that integrated into visual studio then you can perform so many things in visual studio itself like coding, building, packaging, testing, debugging and deploying a driver. You can do large number of debugging tasks on target from host computer like configure a set of target computers for debugging, set breakpoints, execute debugger commands, access memory, access local variables and parameters, remote debugging, user and kernel mode debugging.[1]

2.3 Comparison of all debuggers

Table 2.1: Comparison of all debuggers

Features	**KD/NTKD**	**NTSD/CDB**	**WinDbg**	**Visual Studio**
Kernel mode debugging	Y	N	Y	Y
User mode debugging	N	Y	Y	Y
Remote debugging	Y	Y	Y	Y
Attach to process	Y	Y	Y	Y
Writing extensions	Y	N	Y	N

2.4 Different ways to get debugging tools for windows

There are three different ways to take debugging tools installed in your system for windows platform. Those are listed as following:

- As a part of WDK:
  In this way, two software need to be installed. First you have to install windows driver kit and then download Microsoft visual studio.

- As a part of the Windows SDK:
  In this case you have to install only windows software development kit (SDK). It includes all necessary debugging tools for windows.

- As a standalone tool set:
  In this case you dont need to install other tools that come with SDK. You have to choose only debugging tool at the time of SDK installation. So only debugger will install.

2.5 Debugging modes in Windbg

Windbg can do debugging in both the modes, user mode and kernel mode.

2.5.1 User Mode Debugging

User mode debugging is a simplest form of debugging process which is capable of single target user mode process. In user mode, you can simply examine the program state and you can also modify the states. So it will be notified when special events happening in the target process. Debugger performs debugging process with target debugger till the process is running on target. This is also known as live debugging. There is another concept known as postmortem debugging in which debuggers examine a dump files containing a snapshot of a given process in user mode.
Three built in user mode debuggers come with the windows debugging tools. Those are cdb.exe, ntsd.exe and windbg.exe. These all have same functionality but it works in different ways. These all can do console application debugging and graphical windows program too. If the sources are available than these all can perform source level debugging too. Straight machine level debugging is also done by them.
During debugging in user mode, both the target computer and target application are in frozen mode. When we open debugger in user mode at that time it appears in debugger command window of kernel mode debugger. You can see Input¿ is written

in the lower panel of the Windbg window. After that you can enter any commands related to user mode debugging at this prompt. The debugger will access the symbol files, extension DLLs and such other files which are on the target computer.

2.5.2 Kernel Mode Debugging

Kernel mode debugging is quite complex than user mode debugging. It perform debugging process to analyze computer system as a whole with very nearly same as system processor. In kernel mode, each process and thread is just a collection of data structures. There is a direct relation with the memory addresses with physical memory which is installed on the system. So we cannot access paged out memory without loading it in physical memory.

For kernel mode, live debugging means debugger in kernel mode changes the state of computer and notified such events as well. Device driver developers are using kernel mode debugging. Kernel debuggers are also useful to do user mode debugging in some of scenarios.

Like user mode debuggers, kernel mode debuggers load kernel mode dumps. It analyze offline debugging of an existing system and can also do postmortem analysis of bugs. Two built in kernel mode debuggers come with windows debugging tool. Those are kd.exe and windbg.exe.

During debugging in kernel mode, both target computer and target application are in frozen mode. When we open debugger in kernel mode at that time kd¿ is written in the lower panel of Windbg window. In kernel mode, debugger will automatically detect the platform where target is running.[3]

2.6 Different types to set up kernel mode debugging

You can set kernel mode debugging in two way. First is in visual studio and second is mannualy over different cable/network.[1]

2.6.1 Setting up kernel mode debugging in Visual Studio

Microsoft visual studio can be used to do kernel mode debugging. For that we have to do some setup in visual studio. Kernel mode debugging is only be done by visual studio if visual studio is integrated with windows driver kit (WDK).

2.6.2 Setting up kernel mode debugging manually

Over network cable

Windows XP or later version OS must be there on host computer and windows 8 or later version OS must be there on target computer. Any type of network adapter can be used for host computer. But for target computer, network adapter must be that, that can be supported by windows debugging tools. The following steps are require to setting up kernel mode debugging over network cable:
Step 1: Get IP address of host computer
Use ipconfig command on host computer and get a IPv4 address of the network adapter that you are going to use it. After that you can check it on target by command ping -4 HostName.
Step 2: Select port for network debugging
This port is used by both host and target computer to do the debugging over network. Take any number from 49152 to 65535. Take a number such that same port number is not being used by other process which is running on host computer.
Step 3: Setting up the target computer
Connect target system with supported network adapter. In command prompt win-

dow write bcdedit/debug on. After that write bcdedit/dbgsettings net hostip:w.x.y.z port:n. after that one key is generated automatically. Copy that key as you have to use it in the host computer. After that reboot the target computer.

Step 4: Setting up host computer

Connect host computer with network adapter.

Step 5: Start the debugging session

Open windbg. Go into File menu. In that choose kernel mode debugging. One dialogue box will appear. In that choose net tab. In that enter key which is generated on target and enter port number. After that kernel mode debugging will start.

Advantages of doing debugging on network are as listed below:

- As host and target are connected over network, they can be anywhere on local network. No need to connect physically.
- We can debug many target computer with single host over network very easily.
- Network cables are cheap and easily available.

Over 1394 cable

1394 adapter must be connected with both host and target computer. Windows XP or later version OS must be there on both the system but both host and target should not run on same OS. Both must be running on different OS.

Step 1: Setting up target computer

Whatever 1394 controller you choose, connect it with 1394 cable for both host and target computer. Open command prompt window and write bcdedit/debug on. After that write bcdedit/dbgsettings 1394 channel:n. after that do not reboot the target computer.

Step 2: Start debugging session 1st time

Check 32 or 64bit system is running on host computer. Open windbg for 32/64 bit in admin mode. Open file menu. Choose kernel debug. In that open 1394 tab and write channel number. After that kernel mode debugging will start. For that 1394

debug driver must be installed in your host system. After that reboot target system.
Step 3: start debugging session again
Follow same procedure as given in step 2. You can also start kernel mode debugging by KD or by using environment variables.

Over USB 3.0 cable

To do debugging in this method, a USB 3.0 debug cable and an xHCI host controller on both host and target computer is require.
Step 1: Setting up the target compter
Open USBView tool on target system. Windows debugging tools include this tool. After that locate xHCI host controllers in USBview tool. Expand each node in xHCI host controllers and check the port which supports the debugging. after that note down bus, device and function number of a xHCI controller that you are going to use for debugging. locate physical USB connector with a port on the selected xHCI controller.
Open command prompt window in admin mode and write bcdedit/debug on. After that write bcdedit/dbgsettings usb targetname:TargetName. After that reboot the target system.
Step 2: Start debugging for 1st time
Take USB 3.0 debug cable and connect it with USB 3.0 port which you choose for debugging on both the systems. Check 32/64bit type on host. Open windbg on host. And write target name in the USB tab followed by file menu-¿kernel debug. For this USB driver must be installed in your system.
Step 3: Start debugging again
Follow same procedure as discuss in step 2. After that debugger starts in kernel mode debugging.

Over USB 2.0 cable

To do debugging in this method, a USB 2.0 debug cable and an xHCI host controller on both host and target computer is require.
Step 1: Setting up the target compter
Open USBView tool on target system. Windows debugging tools include this tool. After that locate EHCI host controllers in USBview tool. Expand each node in EHCI host controllers and check the port which supports the debugging. After that note down bus, device and function number of a EHCI controller that you are going to use for debugging. Locate physical USB connector with a port on the selected EHCI controller.
Open command prompt window in admin mode and write bcdedit/debug on. After that write bcdedit/dbgsettings usb targetname:TargetName. After that reboot the target system.
Step 2: Setting up Host computer
Check that host system is not taken as the target of USB debugging. Connect USB 2.0 debug cable with EHCI port on host that does not support debugging or else connect cable with any of EHCI port on host. Connect another end of debug cable with connector on target computer.
Step 3: Start debugging for 1st time
Take USB 3.0 debug cable and connect it with USB 3.0 port which you choose for debugging on both the systems. Check 32/64bit type on host. Open windbg on host. And write target name in the USB tab followed by file menu-¿kernel debug. For this USB driver must be installed in your system.
Step 4: Start debugging again
Follow same procedure as discuss in step 2. After that debugger starts in kernel mode debugging.

Over serial cable

Kernel debugging can also be done over a null modem cable/serial cable in windows debugging tools. You can debug on serial cable by connecting both system with serial ports.
Step 1: Setting up the target computer
Start command prompt window in admin mode and write bcdedit/debug on. After that write bcdedit/dbgsettings serial debugport:n baudrate:rate. The baud rate must be same for both host and target. Reboot the target system.
Step 2: Start debugging
Open windbg in admin mode on host. Open file menu and choose kernel debug. In that choose COM tab. Enter proper baud rate and COM port number that you decide to do debugging on host. After that kernel debugging will start.

Using serial over USB cable

Every platform should not support to open kernel mode debugging using serial over USB cable. Some specific boards only can support it. The other settings steps are as same as we do while using serial cable.

Using a virtual machine cable

When target system is not physical and we use it as virtual then we need to use this method to set up kernel mode. Virtual machine can be on host system itself or it can also be on different system. The all steps to setting kernel mode debugging with virtual machine cable is same as for the serial cable.

Using a single computer

Debugging on local kernel is also possible in windows debug tools. This require only single computer on which debugger runs and same computer is being used for debugging. But in that case break will not work as it would stop OS.

Step 1: Setting up local kernel mode debugging
Open command prompt in admin mode. Write bcdedit/debug on. After that write bcdedit/dbgsetting local. At last reboot your system.
Step 2: Start debugging
Start windbg in admin mode. Go to File menu and select kernel debug. In that choose Local tab. After that you can do kernel mode debugging on Local system.

2.7 Multimedia framework

Mixture of different types of software provided by different sources is called as a Framework. Multimedia framework is flexible and it's architecture is also extensible. Because of such key features, multimedia frameworks allow it's existing services to change as per industry and developers requirements. The developers can achieve the flexibility in framework by using its concept of a component. Each component is simple block and it has only one functionality. Developer uses such different components and builds a more complex system to achieve new feature. Each component has one dedicated API. This API does not give direct permission to specific feature, but developer uses these APIs, assembling it and forming new feature as per requirements. So new developed framework is totally different than those components and their functionality.
In todays market, customers have increased their demands and asking more and more from their multimedia devices. Because of that developers have more pressure to accomplish it.
There are many multimedia frameworks available. Some of popular frameworks are listed below:

- DirectShow
- Ffmpeg

- Gstreamer
- Media Foundation

All listed frameworks are capable to do different functionality. But Gstreamer is a cross platform framework and main advantage of this framework is that it allows pipelining which is not there in any other platform.

2.8 RealSense technology

Intel RealSense technology is providing a platform to users and developers for experiencing amazing things and developing gesture based human interaction techniques accordingly. This technology uses 3D cameras. Intels RealSense Technology has lots of amazing features which are listed below:[6]

2.8.1 Features

- Facial analysis: Tracking multiple faces, Identification of facial features like eyes, mouth, nose
- Hand and finger tracking: tracking 8 different gestures, 10 fingers simultaneously and also allow to access raw depth data
- Sound processing: Speech recognition, speech synthesis, removal of background noise
- Virtual Reality: Object tracking

2.8.2 RealSense 3D Camera

3D camera has made with 4 different components. Those are an IR camera, IR laser projector, conventional camera and a microphone array. By using these components, three different 3D RealSense camera has been developed by Intel. Two are developed for using in tablets and mobile phones while other is developed for using in laptops and notebooks.[6]

- F200 Camera : This is specially designed for front facing analysis.
- R200 Camera : This is specially designed for augmented reality and object scanning. It is rear camera.
- SR300 Camera: This is next generation of front F200 camera.

Chapter 3

WINDBG

3.1 Introduction

Microsoft provides WinDbg debugger for developers. WinDbg called as windows debugger. This debugger can run on the windows based operating system only. It can be used for different purposes. Those are like memory dumps in kernel mode, blue screen debugging, bug check and crash dump in user mode. In some case of debugging, symbol files are needed. So windbg can load symbols in 2 manner, one that manually we can add by using command and second is that automatically windbg can load from a server which matches all criteria. Symbols are two types, private and public. Public symbols are provided by the Microsoft which is available on public server and anyone can use. Public symbols have general .pdb files. Private symbols are very specific to application. It is related to different drivers.
Windbg can do virtual debugging means kernel running on a virtual system. We can run virtual system by using VMware, VPC and paralles by using named pipe. There is virtual COM port available. Debugging over network is also possible. But it can be done on windows 8 and later version of windows OS. It will do fast debugging without any kind of special configuration.
Windbg wraps NTSD and KD with a better UI.

3.2 WinDbg features

3.2.1 Dump files

Dump means you can store some data and then use it for analysis. In terms of debugger, we can take information of a process by dumping the files. You can do mini dump or full memory dump. Information about full threads, their stacks, list of loaded modules are there in the mini dump files. If we do full dump then information about the process heap is also displayed with other information.[2]

3.2.2 Crash dump analysis

Windbg provides the crash dump. In case of windows OS crashes, windbg dumps the information about the crash like physical memory contents. It also displays information of all process which is configured by system-¿control panel-¿advanced-¿startup and recovery. Live process dumping is also possible. That you can do by breaking. If any process terminates abnormally then you can dump that file also. To do crash dump analysis, following steps are needed:[2]
Step 1: Open Windbg. Choose open crash dump in windbg. After that choose the dump file.
Step 2: WIndbg shows the instruction because of what OS crashed.
Step 3: Set proper symbol files that match your process.

3.2.3 Debugger extension DLLs

Windbg allows to use custom commands. Developer create a new commands as per their requirements. Those are known as debugger extensions and written as DLLs. All !(bang) commands are DLL extensions. Load these DLLs in debugger process space.[2]

3.3 WinDbg settings

3.3.1 Symbol files and directories

Symbols are different types. At the time when linker links some application, libraries, OS or drivers, .exe and .dll files are created. At the same time another file also made called symbol files. Symbol files are not useful when .exe files are running. But it is used while debugging. Symbol files contains the following data:

- Global variables
- Local variables
- Line numbers of source code
- Function names and the address of their entry points.
- Frame pointer omission records

Each of above mentioned things are called symbols individually. Symbols can be public or private. Follow the following step to set symbol directories:
Step 1: Open windbg.
Step 2: Choose symbol file path in file menu.
Step 3: Write the path of symbol files.
You can also take symbols from online Microsoft site. It contains some of public symbols on server. We can also set directories by using command .sympath command.[2]

3.3.2 Source code directories

To set source code directories, two methods are available. 1st method steps are as following:
Step 1: Open Windbg.
Step 2: choose Source file option in file menu.

Step 3: Write path where source code is stored.
Second method is directly using .scrpath command in windbg command window.the debugger pulls up source code based on line number information from the symbol files (PDB files) when debugger is running.[2]

3.3.3 Breakpoints and Tracing

You can set soft and hard break points using commands or toolbar options. Dbgprint, KdPrint and OutputDebugString are the tracing routines which print the windbg output window from debugger extension DLLs.[2]

3.4 Types of commands

Windbg have three types of commands.

- Regular (basic) commands
- Dot commands
- Custom commands (DLL extensions)

3.4.1 Regular commands

Regular commands also known as basic commands. They are used to debug process. The regular commands are for different features like Stack trace, Frame, Register watch, Disassemble, Breakpoints, Comment, Continue, Quite, Dumping data, To edit values, List, Modules, Threads, Commands on thread, Dump commands, Searching options, Memory dump, Memory heap

3.4.2 Dot commands

Dot commands are also known as meta commands. They used to control the behavior of the debugger. Some of them are listed below:

- To set symbol path: .sympath
- To reload symbols: .reload
- Some source line related commands

3.4.3 DLL extensions

Windbg allows developer to use custom commands. Developer create a new commands as per their requirements. Those are known as debugger extensions and written as DLLs. All !(bang) commands are DLL extensions. Load these DLLs in debugger process space.
They are implemented as exported functions in forms of extensions DLL. These DLL extensions are user defined commands and they are used to extract specific customized information. You can write any number of new commands in only one DLL extension file. Extension DLL provides extra functionality of automation tasks while performing user mode and kernel mode debugging.[3]

Chapter 4

GStreamer

4.1 Introduction

It is open source multimedia framework. Gstreamer is working on many operating systems like Linux, windows, Android etc. Gstreamer is providing good feature that it can be connect to other multimedia frameworks so that one can reuse existing components like codecs and use pipelining method for input/output mechanisms.[5]

4.2 Features

[5]

- Pipeline structured
- Graphical based construction which makes easier construction
- Object oriented design is possible
- Easy, transparent and simple APIs
- Low latency and high performance
- It has full debugging system

- Core library size is very less as it is in compact format
- It is possible to create multi-threaded pipeline
- Has default plugins
- Capable to extend features by adding new plugins.
- It has development tool and some built in APIs with which developer can directly check their developed pipeline.

4.3 Multimedia technologies supported on Gstreamer

Gstreamer has three different types of plugins. Each contains different types of blocks.
Some of supported formats are listed below which comes with Gstreamer own plugins:[5]

- Audio: multichannel configuration, various bit depths, integer and float types
- Video: interlaced video, progressive video, different color spaces
- Codec: different libraries, ffmpeg libraries, other codec packages from 3rd party
- Container formats: avi, asf, mkv, webcam, mpeg, mov, mp4
- Streaming: rstp, http, mms
- Metadata: native container formats

Chapter 5

Debug Framework

5.1 Generalized block diagram of windows tablet based SoC

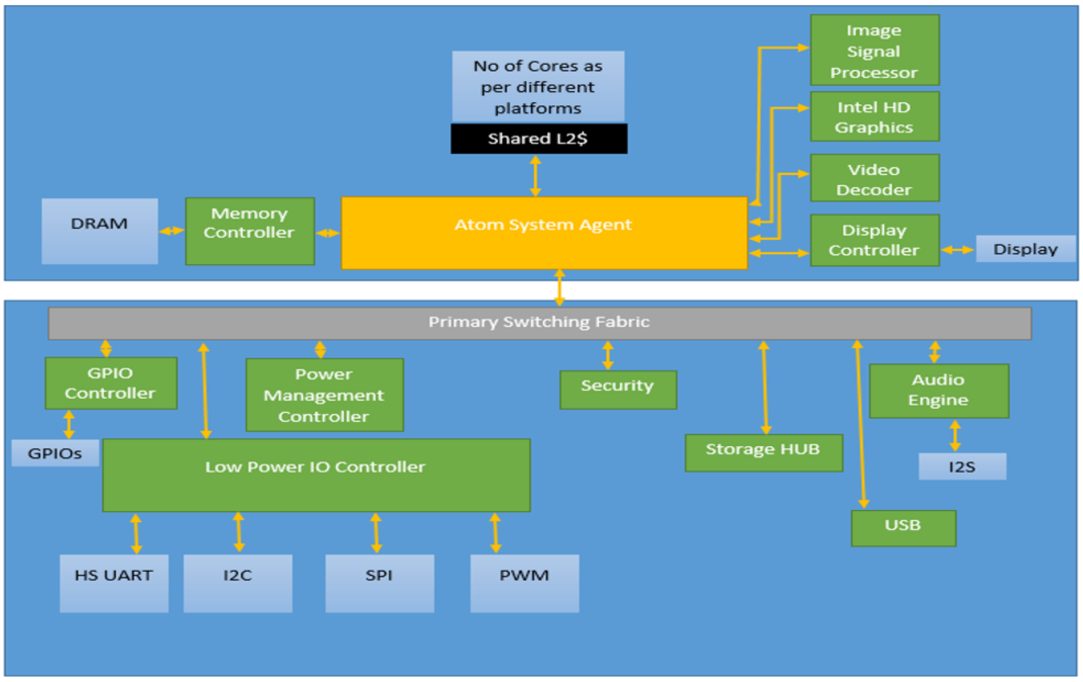


Figure 5.1: Block diagram

5.2 Basic setup flow diagram for starting debugging process

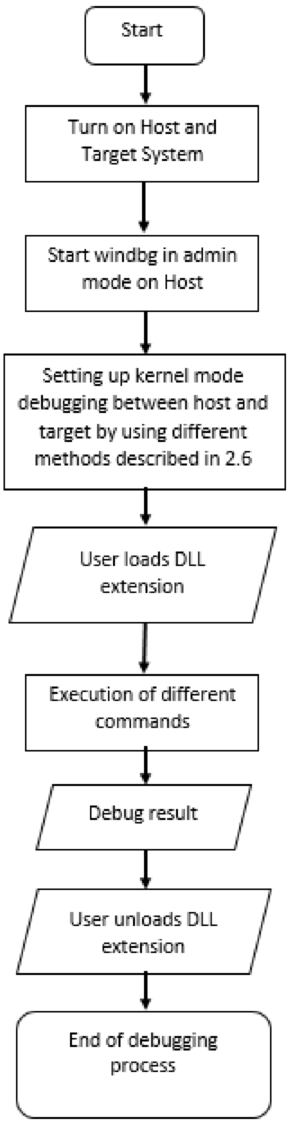
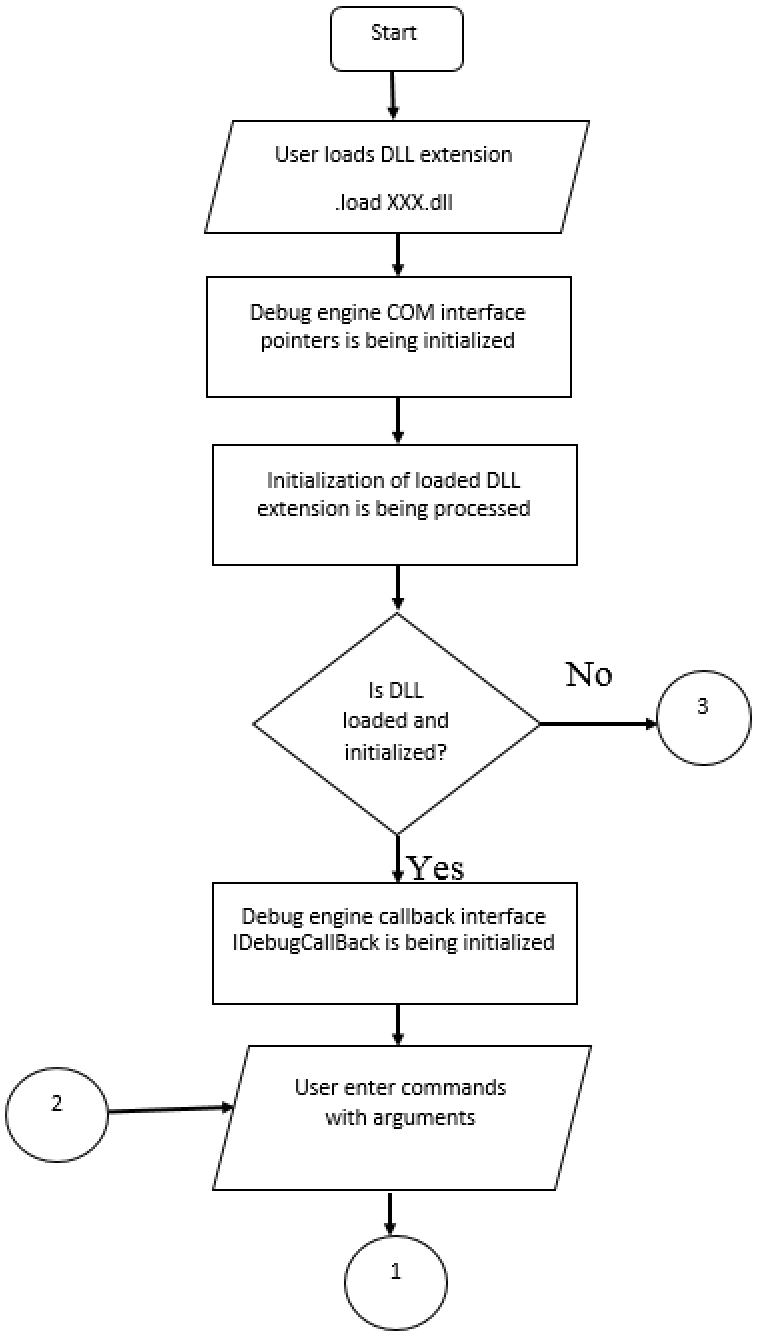

Start
Turn on Host and Target System
Start windbg in admin mode on Host
Setting up kernel mode debugging between host and target by using different methods described in 2.6
User loads DLL extension
Execution of different commands
Debug result
User unloads DLL extension
End of debugging process

Figure 5.2: Setup Flow

5.3 DLL extension program flow in windows


Start
User loads DLL extension
.load XXX.dll
Debug engine COM interface pointers is being initialized
Initialization of loaded DLL extension is being processed
Is DLL loaded and initialized?
No
3
Yes
Debug engine callback interface IDebugCallBack is being initialized
2
User enter commands with arguments
1

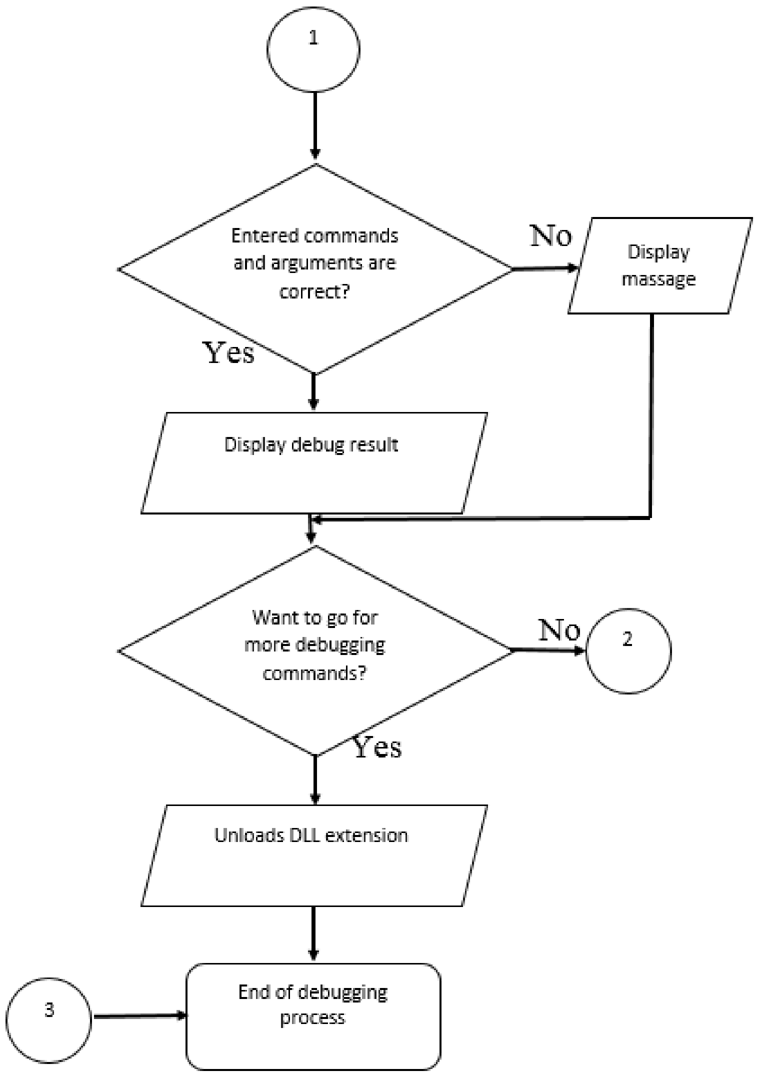


Figure 5.3: Program Flow

5.4 Newly developed DLL extensions

Given basic commands can be used to debug only generalized things. Those things are very basic. To do more specific debugging for advanced platform, new commands are needed. The following all commands are exported in a single .DLL file.

5.4.1 Memory Read/Write

There are two types of memory, physical memory and virtual memory. In the debug process, we have to do read operation directly on some memory location in case of any stuck in process because of that memory value. The same way we have to do write operation on any specific memory location to get out from any memory leak or memory heap.

There are mainly 2 functions for reading memory. Those are described as following:

ReadPhysical(): This function reads directly the targets physical memory. This function has four parameters. One can accept the memory address on which we have to do read operation. The second parameter receives the data that is to be read from that memory address. The third parameter accept the number of bytes to be read from that memory location. The last parameter receives the total number of data to be read from specified memory location. If this parameter is null after the operation then information is not returned from that location.

ReadVirtual(): This function reads directly targets virtual memory. This function also has four parameters and those are as same as in the ReadPhysical() have. The first parameter that accept the memory address that should be in virtual memory address space

There are mainly 2 functions for writing on memory. Those are described as following:

WritePhysical(): This function writess directly the targets physical memory. This function has four parameters. One can accept the memory address on which we have to do write operation. The second parameter receives the data from user that is to be written on that memory address. The third parameter accept the number of bytes to be written on that memory location. The last parameter receives the total number of data to be written on specified memory location. If this parameter is null after the operation then information is not written on that location.

WriteVirtual(): This function writes on directly targets virtual memory. This func-

tion also has four parameters and those are as same as in the WritePhysical() function. The first parameter that accept the memory address that should be in virtual memory address space. These all functions are debug engine interface which are written with IDebugDataSpaces and included in DbgEng.h header file.
The newly developed commands for memory read/write are as following:

- **!memory_read physical_address**: This command has one parameter which accepts direct physical address from user. After that it returns value written on that memory location.
- **!memory_write (physical_address, value)**: This command has two parameters which accepts direct physical address and value to be written on that address from user. After that it writes value on that memory location.

5.4.2 Sideband Port Read/Write

Sideband ports are on the SoC. The difference between sideband ports and IO ports are that IO ports are used to communicate with the peripherals which are outside the SoC while sideband ports are on the SoC only. Sideband ports are used to talk with IPs which are on the SoC. On SoC we have buses through which most of the IPs are connected. Sideband ports are physical ports only. They are not exactly same as IO ports. Most of the sideband ports are memory mapped IOs (MMIO). So we can access those ports directly via accessing main memory by doing proper memory mapping. Mainly two registers are there, control register and data register.
Control Register: This register contain data related port number, which operation you want to do read/write, opcode and such that.
Data Register: This register contains data that is to be written on that sideband port in case of write operation or it returns data that is stored on specified port in case of read operation.
Sideband Read operation:
Step 1: Control register need to be set. So write specific value on control register

as per port number and opcode. For that use memory write function.
Step 2: If offset register address is more than byte, then put MSB byte in extended control register.
Step 3: Read the data from data register. For that use memory read function.
Sideband Write operation:
Step 1: Write the data on data register. For that use memory write function.
Step 2: Control register need to be set. So write specific value on control register as per port number and opcode. For that use memory write function.
Step 3: If offset register address is more than byte, then put MSB byte in extended control register.
Step 4: Again use sideband read function. You can read the value on that port which you wrote on that before using sideband write function.
The newly developed commands for sideband ports read/write are as following:

- **!sideband_read (port address, memory offset)**: This command has two parameters which accepts direct port address and offset value from user. After that it returns value written on that specified location.

- **!sideband_write (port address, memory offset, value)**: This command has three parameters which accepts direct port address, offset and value to be written on that address from user. After that it writes value on that specified location.

5.4.3 IO Port Read/Write

IO ports are used to communicate with input or output peripherals devices which are not included on the SoC. We can access this ports directly in debugging process by sending commands. We can access byte, word or double word on particular IO ports.
There are mainly 2 function which can be used to access IO ports:
ReadIoSpace64(): This function reads the data directly from the specified IO port

location. It has three parameters. One can accept the address of the port. Second parameter contains the address of variable where you want to store the read data. Third parameter takes value in terms of how many bytes you want to read and after operation it will contain how many number of bytes actually read from port.
WriteIoSpace64():This function writes the data directly on the specified IO port location. It has three parameters. One can accept the address of the port. Second parameter contains the address of variable which contains the data to be written on that port. Third parameter takes value in terms of how many bytes you want to write and after operation it will contain how many number of bytes actually written on port.
The newly developed commands for IO port read/write are as following:

- **!io_read_byte port_address:** This command reads byte from the specified IO port. This command has only one parameter that user has to pass at time of debugging. So user has to send port address along with command.
- **!io_read_word port_address:** This command reads word from the specified IO port. This command has only one parameter that user has to pass at time of debugging. So user has to send port address along with command.
- **!io_read_double port_address:** This command reads double from the specified IO port. This command has only one parameter that user has to pass at time of debugging. So user has to send port address along with command.
- **!io_write_byte (port address, value):** This command writes byte on the specified IO port. This command has two parameters that user has to pass at time of debugging. So user has to send port address along with the value which you want write on port with command.
- **!io_write_word (port address, value):** This command writes word on the specified IO port. This command has two parameters that user has to pass

at time of debugging. So user has to send port address along with the value which you want write on port with command.

- **!io_write_double (port address, value):** This command writes double on the specified IO port. This command has two parameters that user has to pass at time of debugging. So user has to send port address along with the value which you want write on port with command.

5.4.4 I2C Read/Write

On new platforms, many of the devices are communicating through I2C buses. There are different number of buses available on SoC. It is depends upon platform and its requirements. I2C is used on single boards. It is also used to connect components which are linked through cables. I2C bus uses only 2 bus line and it can do multi master bus arbitration.
At the time of debugging, we can access the particular I2C and check its status. On windows based tablet platforms, all I2C registers are memory mapped. So we can access it by doing proper memory mapping on main memory by sending base and offset. The following steps are require to do I2C read/write operation.
I2C read operation:
Step 1: Disable bus.
Step 2: Set speed and mode (master or slave)
Step 3: Set target address
Step 4: Enable bus
Step 5: Set read or write mode. Here set control
register in read mode.
Step 6: Read data from data register
I2C write operation:
Step 1: Disable bus.
Step 2: Set speed and mode (master or slave)

Step 3: Set target address

Step 4: Enable bus

Step 5: Set read or write mode. Here set control register in write mode.

Step 6: Write data on data register

The newly developed commands for I2C read/write are as following:

- **!i2c_read (I2C number):** This command has one parameter which is I2C number. For example for i2c0 send argument as I2C0 with command. So that we can see the value on that I2C.
- **!i2c_write (I2C number, value):** This command has two parameters which are I2C number and value that is to be written on that I2C. For example for I2C0 send argument as I2C0 with value. So that we can set the value on that I2C.

5.4.5 GPIO Read/Write

GPIO is general purpose IO pins on SoC. It is generic. User can control it at the time of running. They do not have any predefined functions. On win tablet SoC, there are different GPIO pins. All pins are memory mapped IOs. So those can be access by memory mapped IO operation. All pins have two registers. We can read both of them registers but cannot directly write on every registers. We can write only on some of GPIOs.

The newly developed commands for GPIO read/write are as following:

- **!gpio_read GPIO_number**: This command has only one argument that contains GPIO pin number. This command reads the value of both GPIO configuration register from the entered GPIO number.
- **!gpio_write (GPIO_number,0/1:value):** User has to send three parameters along with command. First command conatin GPIO pin number on which you want to do operation. This command writes the value that is passed in

third parameter to CFG0 or CFG1 as per second parameter we pass. (Second parameter 0 for CFG0 register and 1 for CFG1 register)

5.4.6 PMC related commands

PMC is a power management controller. PMC is very important part on every SoC as it can manage all power related data of each and every devices on that SoC. PMC can do power gated operation. PMC continuously checking power connections and battery charges. It also controls the other ICs power which are connected to each other. It can also perform powering up or powering down of the power domains of the processor.
The newly developed PMC related commands are as following:

- **!pmc_pss_status:** This command gives the power island status of each devices. PSS is stands for power island power states. Power island power states reflects the power status for each physical power island. Power islands can run different functions at different voltages and frequencies at PCB level. So this command reflects that particular device on SoC is power gated for particular power island states or not.

- **!pmc_d_status:** This command gives the D states of devices. Each device can be in any of device states. Device states are categorized in 4 different way:
  D0: fully on state
  D1: Intermediate state
  D2: Intermediate state
  D3: Off state
  At the time of debugging, by using this command we can check the each device D state at any point of time.

- **!pmc_d_status_standby_mode:** This command gives the D states of devices in standby mode. When our system is in standby mode at that time none of devices are in fully active mode. They can be in D1, D2 or D3 states. Standby mode is like sleep state. This command displays in which D state particular device is when whole system is in standby mode.

- **!pmc_s0_time:** This command gives the time spent in S0 mode. Devices can go in different type of sleep states. S0 is like working state. In S0 motion is off but background is on. So this command will give total time spent in S0 mode.
- **!pmc_s0i1_time:** This command gives the time spent in s0i1 state. S0i1 state is state which is mostly same as S0 state only. User is in this idle state when user is interactively using devices.
- **!pmc_s0i2_time:** This command gives the time spent in s0i2 state. S0i2 state is intermediate state.
- **!pmc_s0i3_time:** This command gives the time spent in s0i3 state. S0i3 state is connected standby state.

5.4.7 Reset commands

Reset can clear pending errors and it starts from initial state. Two types of reset is there on most of windows devices, soft reset and hard reset.
Soft reset: It is also known as warm reboot. It is simple reboot process. No power off is occurring during soft reset. When we perform soft reset at that time data is being saved. But the disadvantage of this soft reset is that it cannot release items stuck in memory.
Hard reset: When we perform hard reset at that time complete shutdown will take place. After that again it start to initial state. In hard reset power goes down. It can release items that are stuck in memory.
The newly developed Reset commands are as following:

- **!soft_reset:** This command performs soft reset through windbg.

- **!hard_reset:** This command performs hard reset through windbg.

5.4.8 PUNIT related commands

- **!is_puint_hung:** This command checks either punit is in hung mode or not
- **!punit_info:** This command gives punit version.

5.5 List Of New Commands

Table 5.1: List Of all commands

Commands	**Parameters**	**Description**
!memory_read	address	reads physical memory
!memory_write	address,value	writes on physical memory
!sideband_read	port,offset	reads sideband port
!sideband_write	port,offset,value	writes on sideband port
!io_read_byte	port address	reads byte from IO port
!io_read_word	port address	reads word from IO port
!io_read_double	port address	reads double word from IO port
!io_read_byte	port address,value	writes byte on IO port
!io_read_word	port address,value	writes word on IO port
!io_read_double	port address,value	writes double word on IO port
!i2c_read	i2c number	reads from I2C
!i2c_write	i2c number,value	writes on I2C
!gpio_read	GPIO pin	reads from GPIO pin
!gpio_write	GPIO pin,select reg,value	writes to GPIO pin
!pmc_pss_status	-	power island status of device
!pmc_d_status	-	D state of device
!pmc_d_status_standby_mode	-	D state of device in standby mode
!pmc_s0_time	-	time spent in s0
!pmc_s0i1_time	-	time spent in s0i1
!pmc_s0i2_time	-	time spent in s0i2
!pmc_s0i3_time	-	time spent in s0i3
!soft_reset	-	soft reset
!hard_reset	-	hard reset
!is_puint_hung	-	PUNIT is hunging or not
!punit_info	-	PUNIT version

Chapter 6

Debug utility

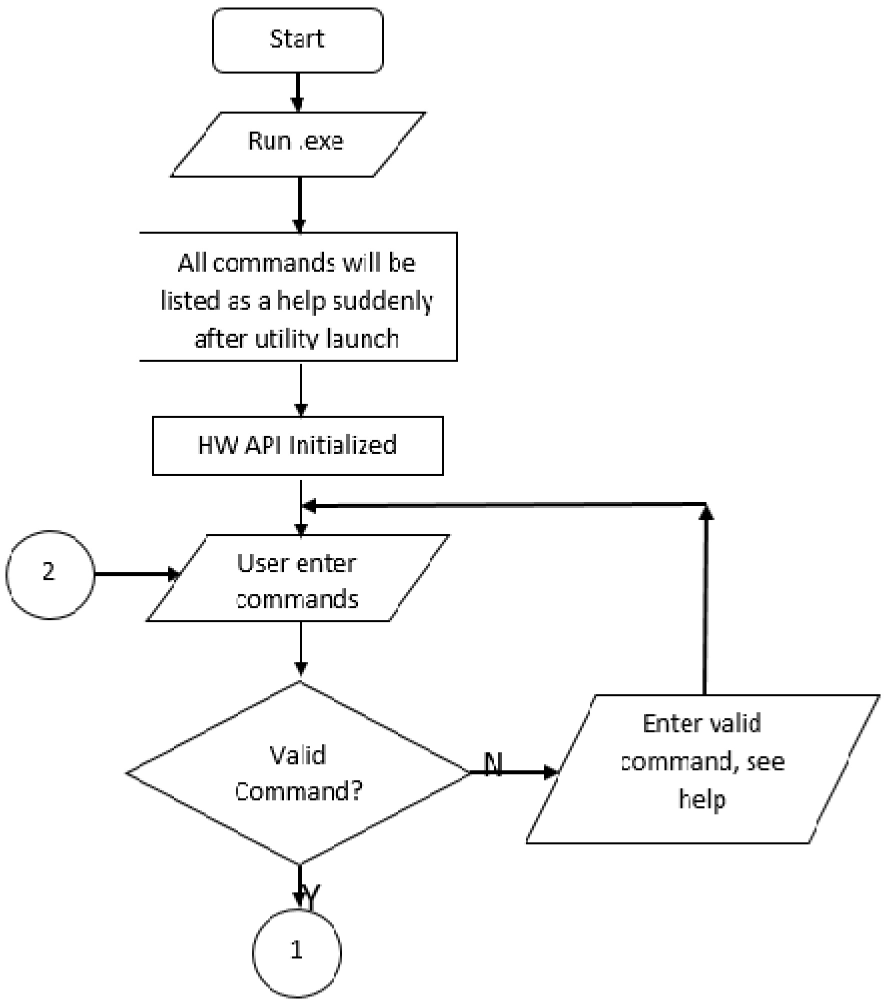
6.1 Features

This utility is developed to debug target system by running it directly on the targeted system instead of host. This tool does not have any dependency as it is standalone utility. It is aimed to develop commands that can check each device status on target. All DLL extension plugins listed in previous chapter are developed in this utility. All functionality are exactly as same as plugins listed in previous chap. The only difference is that all those commands run from windbg which is windows debugger and can be run only from host system. On the other part this debug utility is running directly on target so all these commands run on target directly.
The list of commands included in utility are:

- memory_read/memory_write
- io_read/io_write
- sideband_read/sideband_write
- gpio_read/gpio_write
- pss_status

- D_status
- D_status_standby_mode
- S0_time
- S0i1_time
- S0i3_time
- warm_reset
- cold_reset
- i2c_read/i2c_write

6.2 Utility internal flow for starting debugging process directly on target


Start
Run .exe
All commands will be listed as a help suddenly after utility launch
HW API Initialized
2
User enter commands
Valid Command?
N
Enter valid command, see help
Y
1

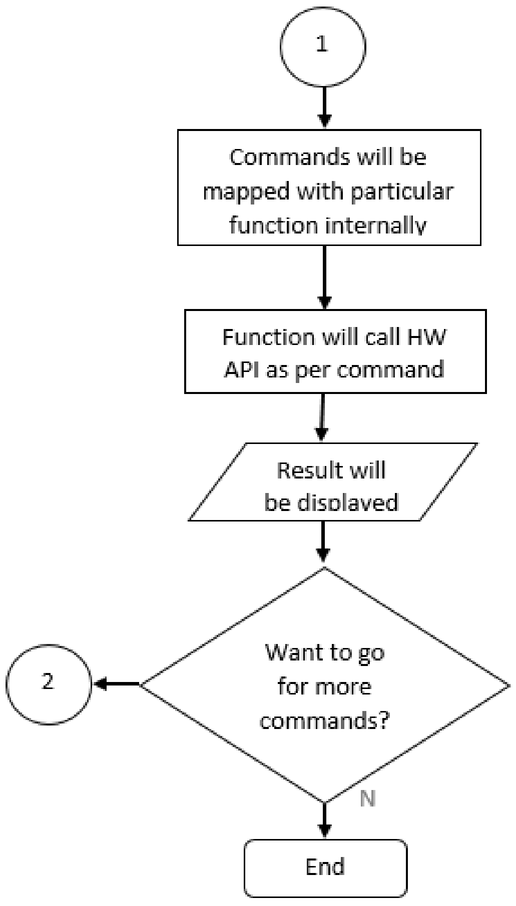

1
Commands will be mapped with particular function internally
Function will call HW API as per command
Result will be displayed
Want to go for more commands?
2
N
End

Figure 6.1: Utility Internal Flow

6.3 Impact of this tool in debug environment

- When we do debugging from host system, we need mechanism in terms of both HW and SW. This is not needed in this utility.
- Dependencies are removed
- When we do debug from host at that time target system will go in halt state so we cannot debug some running scenario data which it is not there in this utility.

Chapter 7

Energy monitoring tool

7.1 Overview

Traditionally the power management and OS configuration settings were done via firmware interfaces. In that, to do any change every time we need to change BIOS settings. And all this process is through firmware not via OS which in turn restrict some functionality. It was very much difficult to code in BIOS for power management. It was also restricted to hardware static configuration.
Because of all these limitations Advanced Configuration and Power Interface (ACPI) has been developed. ACPI is an interface between OS and hardware to overcome above discussed limitations. By using ACPI, we can do power management directly through OS without doing any change in BIOS.
Through ACPI we can get information about power states, device states, battery related information, thermal related information and such power related things. To get all such information while debugging, every time debuggers are checking ACPI table. In ACPI Table, they are reading values on specific offset for specific data. So debuggers need offset values for each specification.
To make this debugging process smarter and easier, this Energy Management Tool has been developed. This tool is showing all necessary information related to battery,

thermal, EC and energy related platform status.

7.2 Communication through EC

Debuggers can query the battery, charging, thermal energy and temperature status through embedded controller (EC). At every platform, some private ports are provided to communicate through EC for getting queries. This communication can also be done via other command ports provided for BIOS. But in that we will get OS-EC traffic and we may get incorrect data or time out error. To prevent such error its better to use private port which specially associated for EC only.
The functionality of 2 ports are:
Command/Status Port: This port is dedicated for sending commands and reading status.
Data Port: This port is dedicated for getting data.

7.3 Features

7.3.1 Power Source

Table 7.1: List Of power source related features

AC adapter status	showing whether AC source is available or not
Power button status	Power is ON/OFF
System power	showing power available on system in mWatt

7.3.2 Battery information

Table 7.2: List Of battery related features

Design capacity	showing design capacity of battery in mAh
Full charge capacity	showing battery's full charging capacity in mAh
Remaining capacity	showing battery's remaining capacity in mAh
Battery voltage	showing battery voltage in mVolt
Available battery percentage	showing available battery

7.3.3 Charging Information

Table 7.3: List Of charging related features

Charging Status	showing battery is charging or discharging or connected or disconnected
Charge Current Limit	showing charging current upper limit in mA
Charging current	showing current at time of charging in mA

7.3.4 Platform Status

Table 7.4: List Of platform related features

EC Version	showing EC version
Board ID	showing board ID
Fab ID	showing silicon ID
Lid Switch status	showing that lid switch is enabled or disabled
Virtual Battery status	showing that virtual battery is enabled or disabled
Fan Speed	showing fan speed in rpm
Type C USB	showing type C USB is Plugged or Unplugged

7.3.5 Thermal Information

Table 7.5: List Of thermal related features

DPTF Charge Current	showing DPTF charging current in mA
Thermal Sensors Temp	showing of all available thermal sensors on system in degree Celsius
CPU Temp	showing CPU temperature

7.3.6 Log Files

Table 7.6: List Of generated log files

logs.txt	all values are logged in this text file after some decided delay
errors.txt	if any error generate then it will be stored in this text file

7.4 Detailed internal flow diagram

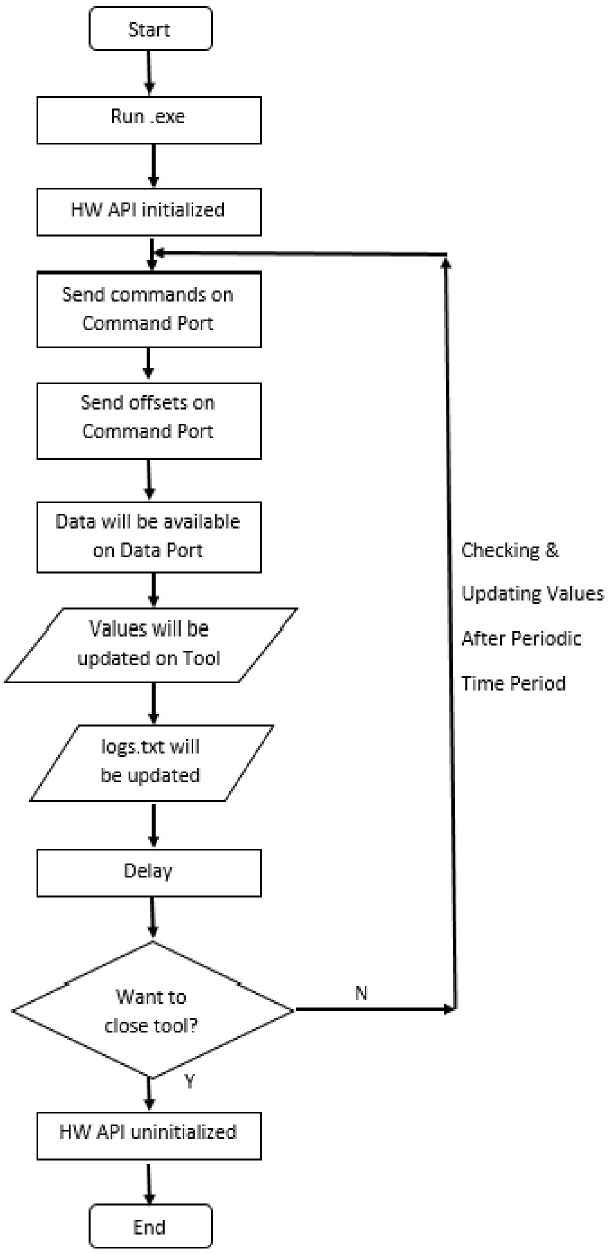

Start
Run .exe
HW API initialized
Send commands on Command Port
Send offsets on Command Port
Data will be available on Data Port
Values will be updated on Tool
logs.txt will be updated
Delay
Want to close tool?
N
Y
HW API uninitialized
End
Checking & Updating Values After Periodic Time Period

Figure 7.1: Internal Logic Flow

Chapter 8

Bi camera capturing in a single window with single encoded file

8.1 Overview

There are many applications in which dual mode camera previewing is possible. In many of video calling applications this dual mode camera previewing feature is implemented. But in all those applications, both camera, front and rear are being previewed on a different window. On the other side, not a single application is there to encode both camera in a single file. We can get either previewing or encoding at a time.

This application is aimed to preview both camera at same time in a single window as picture in picture format and simultaneously at backend it also create single encoded file. These both feature are implemented in single application.

Some windows based multimedia frameworks are available openly. To fulfill this requirement, as per my literature study gstreamer is best suited framework. Because it allows developers to create their own pipeline as per their need. This tool has been implemented using gstreamer plugins. In detail it is discussed in next sections in this chapter.

8.2 Feature

As shown in figure, need is to preview dual camera video in a single window and at end getting single encoded file which contains both camera video in same format as it is previewing.

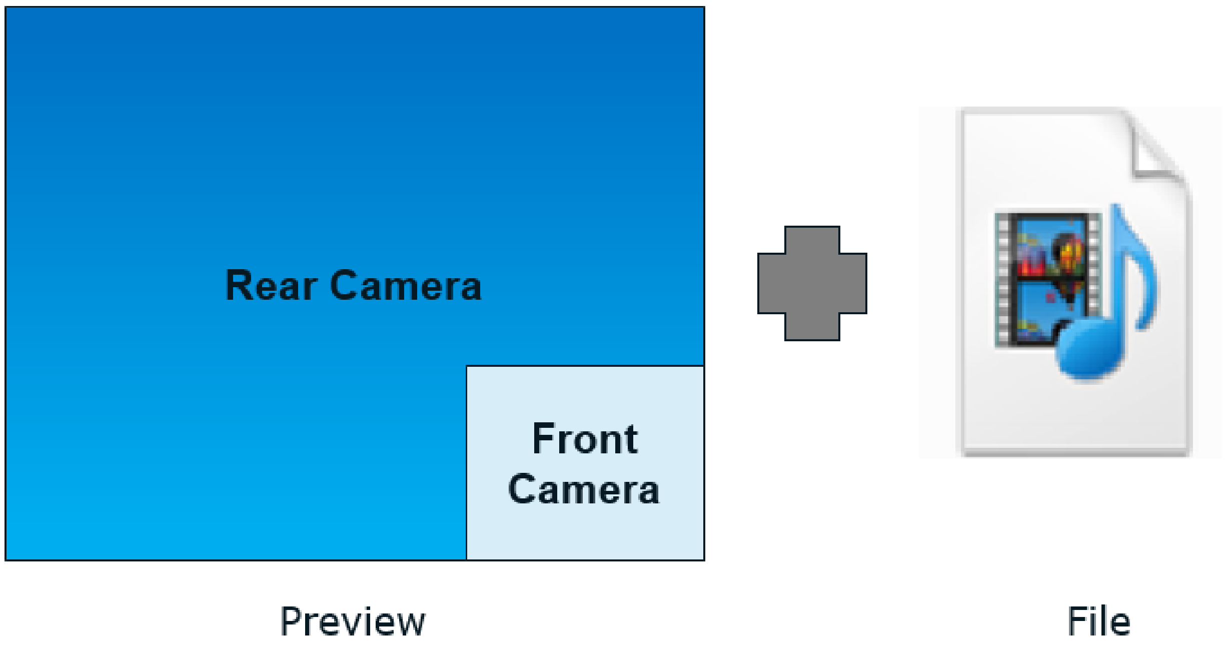


Figure 8.1: Requirement

8.3 Implementation

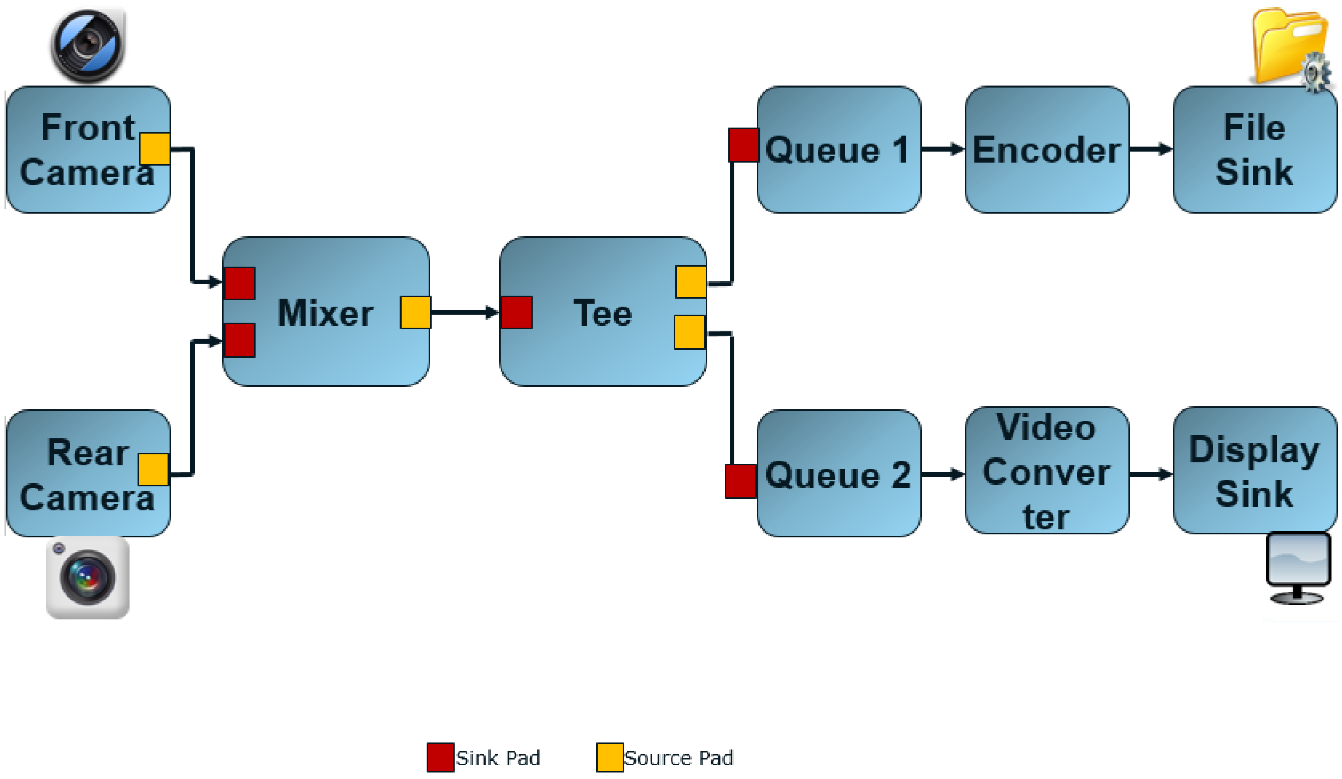


Figure 8.2: Implemented Pipeline

8.3.1 Description of each block

Video Source:

- Source 1: Front camera is one video source. We need its preview in small portion over the main camera preview.
- Source 2: Rear camera is another source. This preview we need as a main source on a big picture window.

The raw stream coming from both camera source will be stored in sink pad. These both sink pads are connected with mixer's source pads.

Mixer:

This block will mix the 2 raw streams coming from 2 different sources in picture in picture format. After that it will be stored in mixer's sink pad.

Tee:

This block can create 2 copy of streams from original mixer stream and send it to the sink pads. Tee's sink pads are connected with 2 different queues.

Queue:

- Queue 1: This queue takes the raw stream from tee and send it to the encoder. It acts as a simple buffer.
- Queue 2: This queue take another copy of mixer stream and send it to the video converter block to display it on screen. This queue is also work as a simple buffer.

Encoder:

This encodes raw video stream in a decided format.

Video Converter:

This block simply convert the raw stream in a specified format just for displaying.

Sink:

- File Sink: This sink stores the final single encoded file. This file we can save it and see it as a normal video.
- Display Sink: This sink displays the both camera preview as a picture in picture format.

Chapter 9

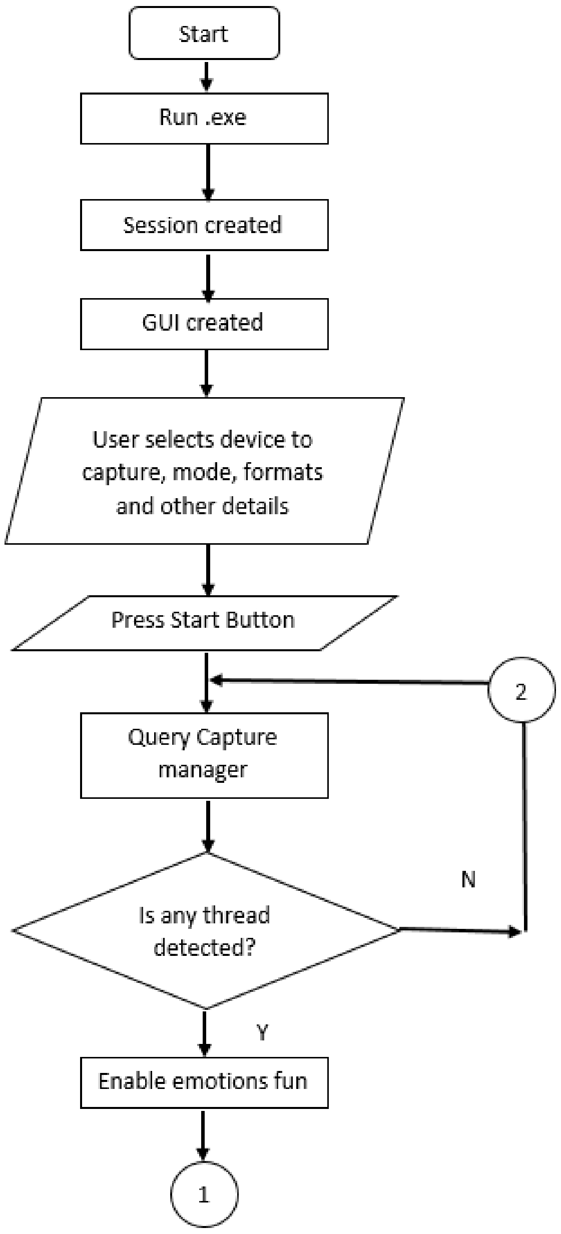
RealSense apps

9.1 Emotion detection

9.1.1 Functionality

Emotion detection app is aimed to detect human's emotions in real life. This app recognize as many number of faces in front of 3D camera and detecting emotions for every faces. The detection of emotion is possible as fast as human's emotion change. This app also replace the face of human with according emojis.

9.1.2 Implementation



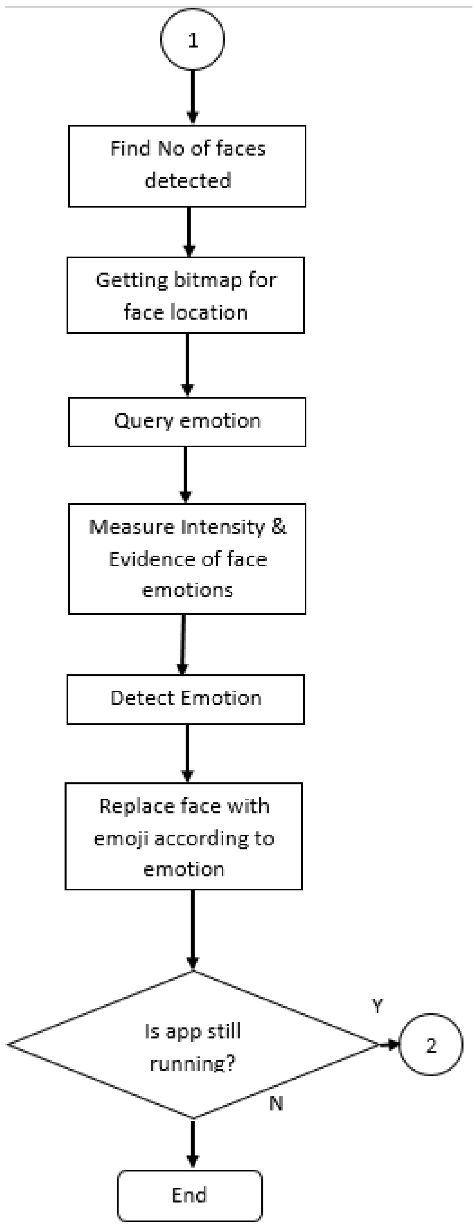
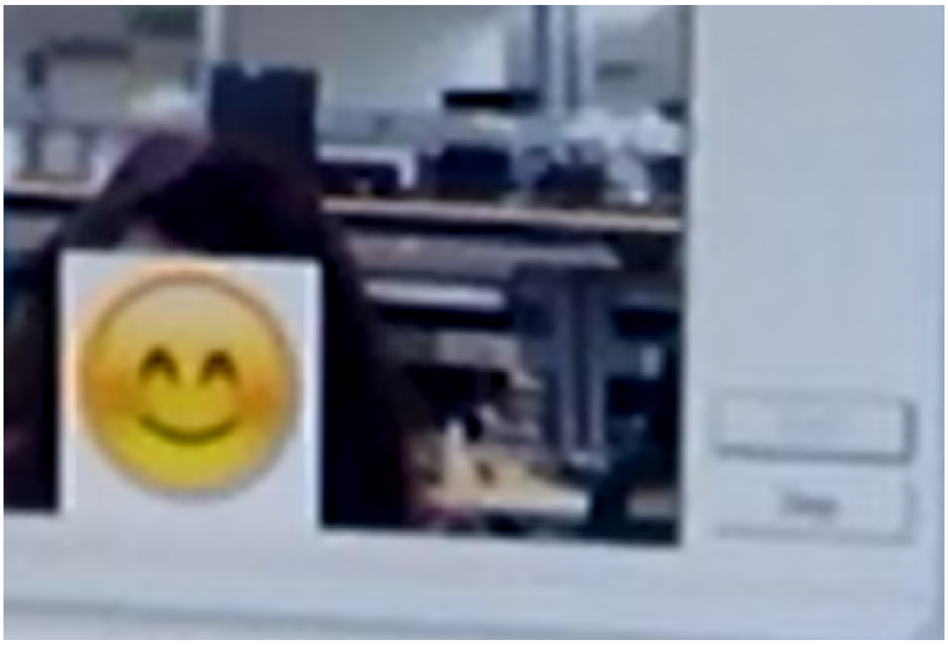

1
Find No of faces detected
Getting bitmap for face location
Query emotion
Measure Intensity & Evidence of face emotions
Detect Emotion
Replace face with emoji according to emotion
Is app still running?
Y
2
N
End

Figure 9.1: Logic flow for emotion detection

9.1.3 Result



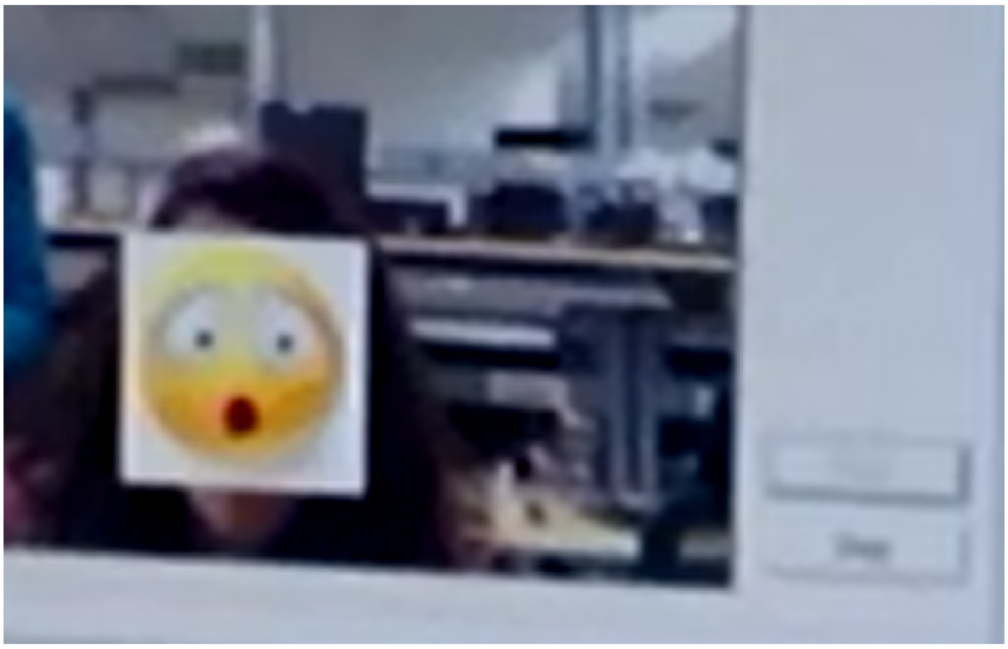


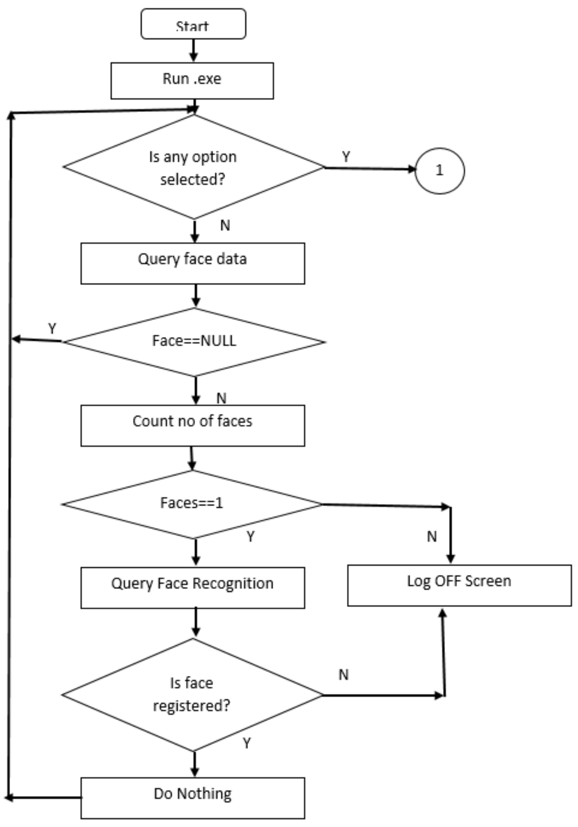
Figure 9.2: Snapshots of app1

9.2 Secure system

9.2.1 Functionality

Today everyone need to secure their confidential data and work at either corporate level or private level. Everyone need to secure their system either in their absence or in present. So this app is aimed to secure the individuals system.
This app run continuously on windows based system behind other running applications. It recognize only authorized persons. if any unregistered person come in front of our system and watching in our system without our knowing then it will directly log off our screen without asking our permission.
This app has a feature to register or unregister user, save or delete database.

9.2.2 Implementation



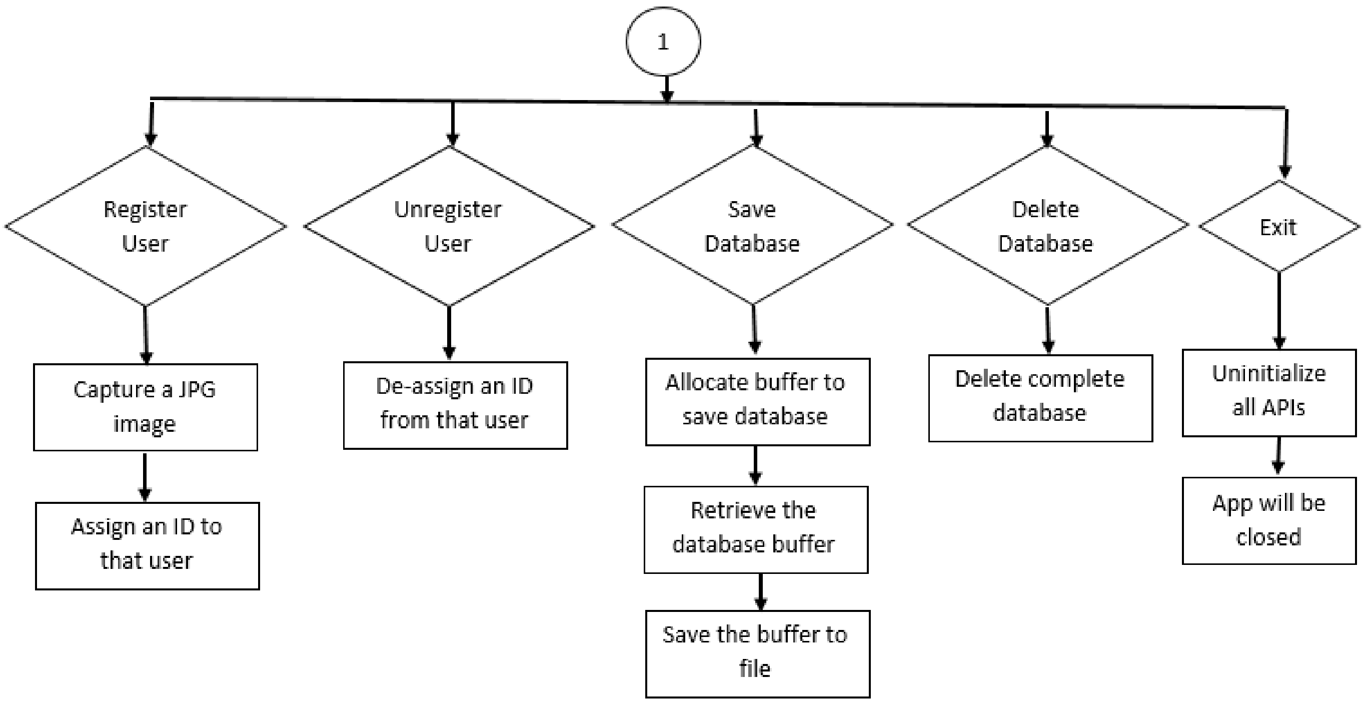


Figure 9.3: Logic flow for secure system

9.2.3 Result

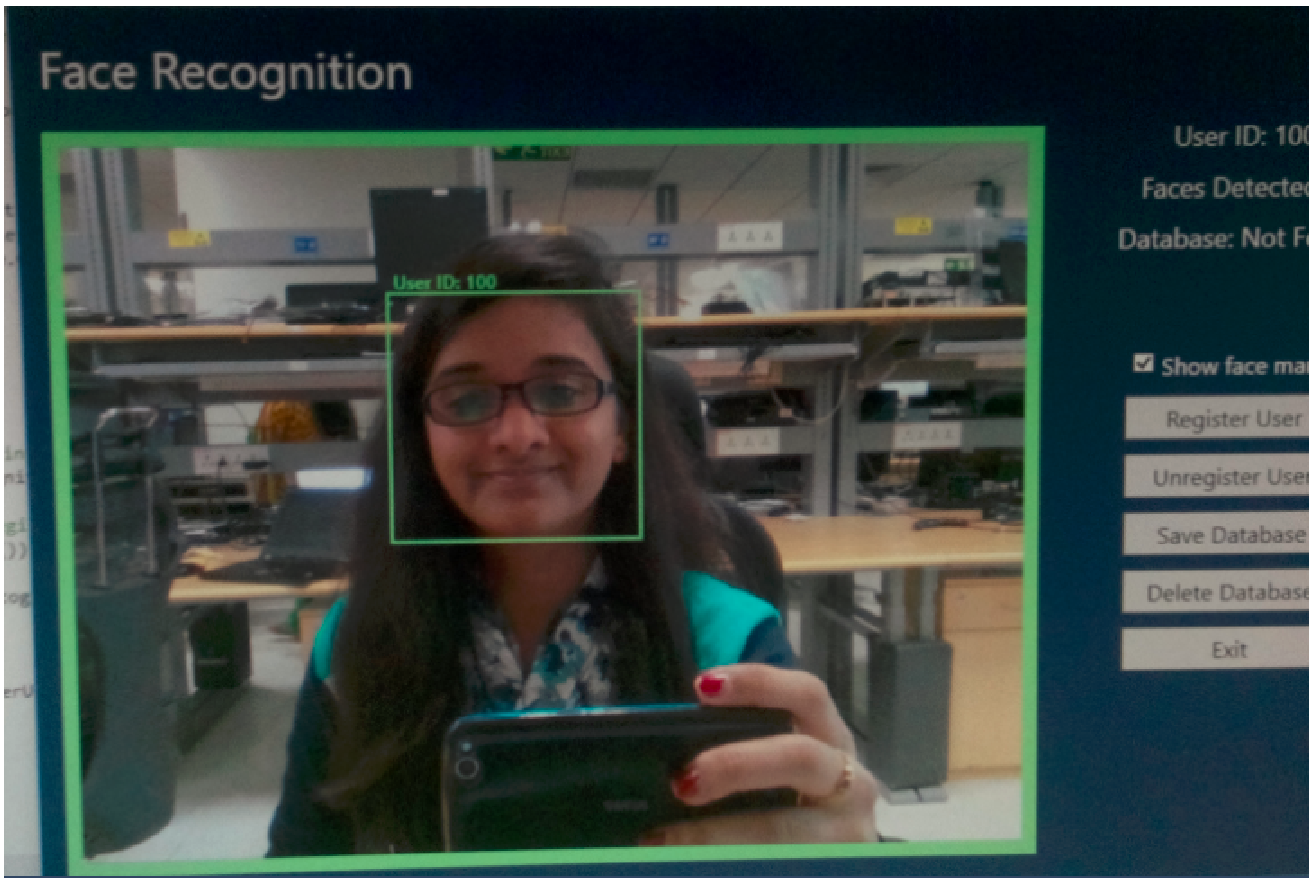


Figure 9.4: Snapshots of app2

Chapter 10

Conclusion

All debug tools and plugins developed in this project would help to carry out a thorough analysis of upcoming technologies. These all tools have eliminated the need of recalling address and other platform specific details. These tools have reduced the time of debugging process and enabled smarter process.
The multimedia apps developed in this project would give new exciting features by using latest technologies and give amazing experience to users by enabling smarter features. Along with that, it will be used by debuggers also to debug those new features related test cases.

Chapter 11

Future scope

In this project, different tools and plugins are developed for different features. We can make a single generic debug tool in which all feature will be covered and can be used for all platforms.
The connection procedure of host with target system at is some what tedious task. So we can write automation scripts to setup host and target connection.
The developed real sense apps in this project can be synced with default skype/lync application. So that users can experience new amazing features and also secure their system.

Bibliography

[1] "Windows Debugging Tools and Setup",[Online],Website,July 2015, $https : //msdn.microsoft.com/en{-}us/library/windows/hardware/ff551063(v = vs.85).aspx$

[2] "WinDbg Details",[Online],Website,July 2015, $http : //www.codeproject.com/Articles/6084/Windows - Debuggers - Part - A - WinDbg - Tutorial$

[3] Mario Hewardt and Daneil Pravant, *Advanced Windows Debugging*. The Addison-Wesley, Microsoft Technology Series

[4] Intel internal documents

[5] "GStreamer Details",[Online],Website,Feb 2016, $http : //gstreamer.com$

[6] *Intel RealSense SDK Documentation*. By, Intel Technology Series