DESIGN OF ENERGY EFFICIENT WEARABLE ECG SYSTEM AND LOW POWER ASYNCHRONOUS MICROCONTROLLER

ZHANG DA REN

NATIONAL UNIVERSITY OF SINGAPORE

2012

Acknowledgements

First of all, I would like to thank my supervisors Prof. Lian Yong for his encouragement and advice during my Master study. His guidance helps me a lot through this work.

Secondly, I am grateful to my project team members, Mr. Xu Xiao Yuan, Chacko John Deepu and Yang Tao for their continuous work and help on wearable ECG system; and Mr. Xue Chao for his explaining of Asynchronous microcontroller part.

Thirdly, I would like to thank Mr. Teo Seow Miang and Ms. Zheng Huan Qun for their technical support. My appreciation also goes to all my colleagues and friends of the Signal Processing & VLSI lab. They are Zhang Jinghua, Zou Xiao Dan, Tan Jun, Liew Wensin, Niu Tian Fang, Zhang Xiao Yang, Wang Lei, Zhang Zhe, Li Yong Fu, Hong Yi Bin, Chen Xiaolei, Yang Zhenglin, Li Ti, Yu Heng and many others.

Lastly, but most importantly, I would like to dedicate this thesis to my beloved parents Zhang Bao Chen and Xing Bin Wa. Their continuous encouragement and support always give me confidence through my life.

Contents

Summary

This work is about the design and implementation of energy efficient wearable real time monitoring ECG system and a low power asynchronous 8051 microcontroller for biomedical sensor interface device. It is motivated by the increasing awareness of Cardiac arrhythmias and coronary heart disease due to population ageing and stressful modern life.

The hardware, firmware and graphical user interface are developed for energy efficient wearable ECG system. There are two designs of wearable ECG system in this work. The first design is a Wireless ECG Plaster prototype device. It is designed for real-time monitoring of ECG in cardiac patients. The proposed device is light weight (25 grams), easily wearable and can wirelessly transmit the patient's ECG signal to PC using ZigBee. The device has a battery life of around 26 hours while in continuous operation, owing to a low power BMDAV8 ECG acquisition front end chip. The prototype has been verified in clinical trials and variation is very low at 0.4% compared to the reference device. The second design is a Long Playing Cardio Recorder system prototype. It is designed for 48 day long term ECG data recording, and it is also a wearable device. It receives data from an ultra-low power ECG acquisition chip. The data is stored into a 16G bit NAND flash. The system current consumption could be less than 1.7mA from a 3.7V 650mAH Li-ion battery so it can

last for 30 days.

To further reduce the power consumption for wearable ECG system, a new design of 3.3V to 1.0V voltage-scalable asynchronous 8051 Microcontroller is presented. The asynchronous core of the proposed design is synthesized in the Balsa framework using the dual-rail four-phase approach. With the same synchronous 8051 microcontroller instruction set which includes add, jump, and multiply operations verified in simulation, the proposed AMS 0.35μm technology microcontroller consumes about 40 μW at 1.0V supply.

List of Tables

List of Figures

List of Abbreviations

A/D	Analog-to-digital
ADC	Analog-to-digital converter
AF	Aritrial fibrilation
ALU	Arithmetic and Logical Unit
AMS	AustriaMicroSystem
BPM	Bit per minute
CHD	Coronary heart disease
CMOS	Complementary metal-oxide-semiconductor
CISC	Complex Instruction Set Controller
DAC	Digital-to-analog converter
DRL	Right-leg driver
DSP	Digital signal processor
ECG	Electrocardiogram
EDA	Electronic design automation
HDL	Hardware description laguage
GDS	Graphical Database System
IF & ID	Instruction Fetch and Instruction Decoding
LEF	Libraray exchange file
LHP	Left-half-plane

LPCR	Long time cardio recording
LPE	layout parasitic extraction
MIP	Million instruction per second
P&R	Placement and routing
ROM	Read only memory
S/H	Sample-and-hold
SOC	Silicaon on chip
SPICE	Simulation Program with Integrated Circuit Emphasis
USB	Universal Serial Bus
VLSI	Very Large Scale Integration

Chapter 1

Introduction

Cardiac arrhythmias and coronary heart disease (CHD) constitute significant public health burdens. Researches show that US$173 billion is spent every year for treatment of heart related disorders in USA [1]. Atrial fibrillation (AF), a common arrhythmia, afflicts nearly 9% of persons over 80 years old [2], and is associated with increased stroke risk. Another arrhythmia, ventricular arrhythmia, can cause sudden cardiac arrest. For heart related disorders, the chances of a total and fast recovery of the patient are diminished by the late detection of the symptoms, which may cost patient's life. Early diagnosis presents an opportunity for preventive treatment. However, many patients with cardiac arrhythmia or silent myocardial ischemia remain undiagnosed and untreated, because abnormal electrocardiogram (ECG) changes often occur sporadically and are easily missed. Hence, a better ECG monitoring device is necessary.

In recent years, personal ECG monitoring medical device has attracted increasing

attention as it reveals to be a promising solution to the overwhelming demand in healthcare industry due to population ageing. There are hundreds of portable ECG monitoring systems in this market. The commonly used solutions like ambulatory Holter systems are often bulky with many wires stuck on patient's chest. The operational life of the Holter is usually limited within 24 hours, and ECG data are analyzed offline for diagnosis of the problem. One major shortcoming of the existing ambulatory Holter systems is extremely low diagnostic yield at 10-13% [3]. In addition, such devices are quite heavy and use traditional ECG electrodes, which are not comfortable as there are multiple wires hanging over the body. And such devices usually aren't waterproof; therefore, the patient is expected to avoid water contact in the area where the device is fixed. All these compromises patient's comfort level and affects his life style.

To avoid the limitations of such a kind of Holter device, the motivation of this work is to present energy efficient wearable ECG monitoring system. There are two phases for this work. In the first phase, a wireless ECG plaster prototype device is designed for real-time monitoring of ECG in cardiac patients. This device, when placed on patient's chest, continually records single-lead ECG and wirelessly streams it to a remote station for diagnosis. The skin contact electrodes have been printed on flexible substrates with consideration for easy wearability. A highly integrated, low power chip with low noise amplifier, ADC and low pass filters were developed in-order to reduce the power consumption and the number of discrete IC components.

In the second phase, another ECG monitoring device, Long Playing Cardio

Recording Version 1 (LPCRV1) system is designed. It can store 48 days ECG data. It is designed for special requirement of long time ECG recording. The system still keeps the advantage of light weighted and smaller in size from Wireless ECG Plaster. Its firmware can maintain ultra low power consumption when huge data reading and writing in order for long term used. The version 1 is the first version of Long Playing Cardio Recording system. In this version, device uses traditional ECG lead contacts to collect ECG signal instead of comfortable substrate. The focus of this version is low power, long time playing and large ECG data recording in NAND Flash.

In addition, the microcontroller is a significant source of power consumption unit. In order to further reduce the power consumption of the wearable ECG monitoring system above, a microcontroller which consumes less power is desired. Therefore, this work also aims to design a new version of low-power asynchronous 8051 microcontroller based on previous work. This microcontroller works as a local processing and control unit in a bio-medical sensor interface block which is powered by batteries. It follows the structure of a standard synchronous 8051 microcontroller invented by Intel, so firmware developer can use it easily. The asynchronous core of the proposed design is synthesized in the Balsa framework using the dual-rail four-phase approach. Furthermore, the core's structure adopts No pipeline structure together with Multiplication and Division block to improve power performance of asynchronous 8051 microcontroller.

The organization of this dissertation is as follows. In Chapter 1, introduction and motivation of this work is introduced. Chapter 2 outlines a brief background of the ECG and asynchronous circuit design. Chapter 3 and 4 elaborates Wireless ECG

Plaster and Long Playing Cardio Recording system individually. In Chapters 5, the wearable ECG system comparison will do some performance analysis here. Chapter 6 details a new design for low power asynchronous 8051 microcontroller which is designed for further reduce the power consumption of wearable ECG system in the future. Chapter 7 concludes the work.

The Wireless ECG plaster of this work was accepted by the Biomedical Circuits and Systems Conference (BioCAS), 2011 [4].

Chapter 2

Background

2.1 Wearable ECG system

2.1.1 ECG introduction

Electrocardiography (ECG) is an interpretation of the electricity activity of the heart over a period of time, as detected by electrodes attached to the outer surface of the skin and recorded by a device external to the body. Generally speaking, the ECG signal shown in Figure 2.1 can reflect the electrical activities of a person's heart over time. Not only does it reflect his or her heartbeat, but also it provides greater insight to the detailed biological activities of the heart. Because it can be obtained through simple and nonintrusive procedures, the ECG signal has been one of the most sophistically studied and widely used indicators for diagnosing heart diseases.

Based on the early studies on dogs in the 1950s and the latter similar studies on the human heart in the 1970s, it is commonly accepted that the ECG signal is essentially generated from the propagation of dipole wave fronts across the heart tissue that originate from the depolarization and repolarization processes in the heart

cells.

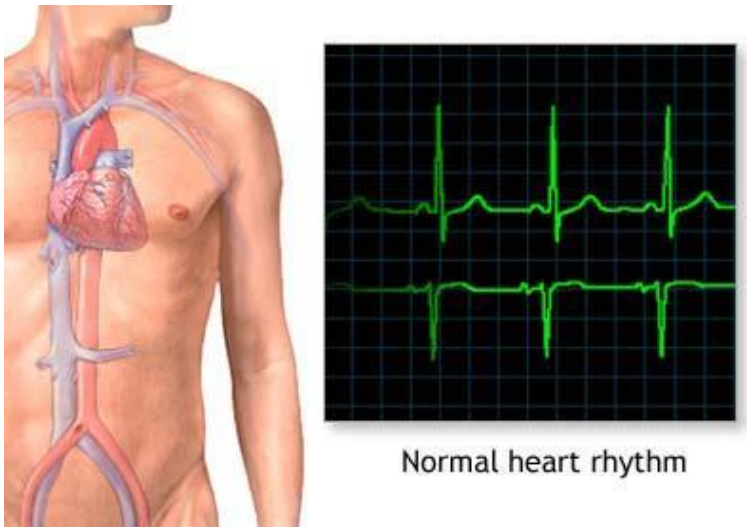


Figure 2.1: The ECG signal

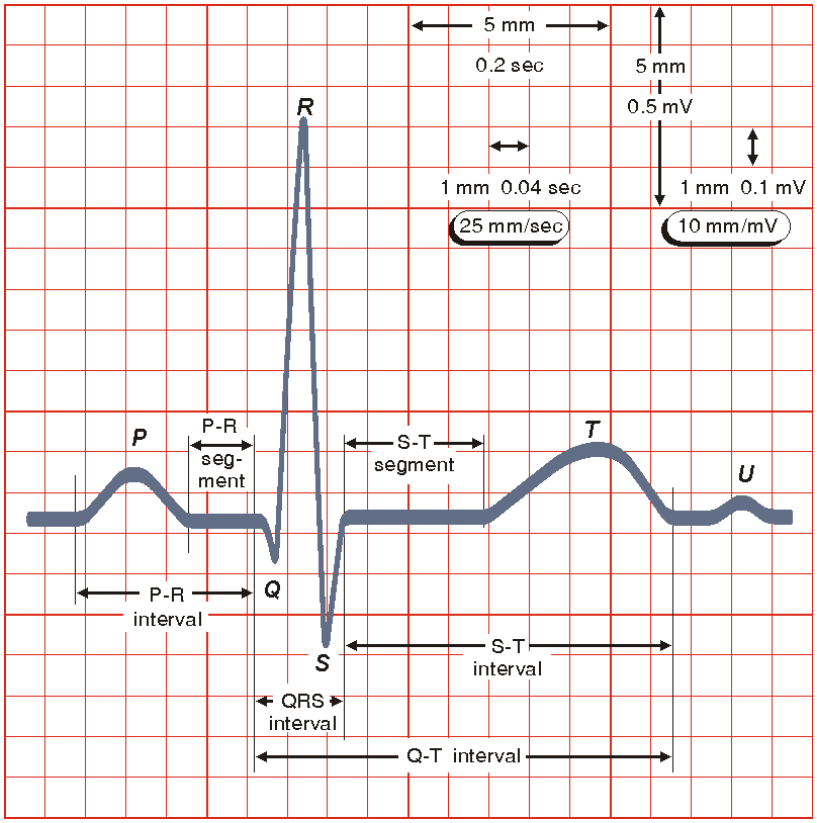


Figure 2.2: The normal ECG signal in one cardiac cycle.

Figure 2.2 depicts one cycle of the typical ECG signal obtained and recorded on the standard ECG paper. The deflections are named in alphabetic order as *P* wave, *QRS* complex, *T* wave and *U* wave respectively. The various segments and intervals are defined and used extensively in diagnoses.

The *P* wave corresponds to the atrial depolarization. The ventricular depolarization occurs during the *QRS* complex. The repolarization of the atria also takes place in this interval but is too small to be observed in the ECG. The *T* wave forms when the ventricles repolarize from activation. The formation of the *U* wave is not very clear yet, and it is normally seen in 50% to 75% of ECGs [5].

2.1.2 ECG monitoring system Literature Review

ECG monitoring system is for monitoring patient's ECG status and recording the data. The basic requirement for telemetric ECG recording system, especially for a portable/wearable one, is ultra-low power consumption. The ultra slim rechargeable batteries manufactured for good portability today usually have only a few hundred mAh of capacity. To operate the ECG device for weeks, the average current consumption thereby should be strictly controlled within mA range. Because the majority of the current has to go to the telemetry or storage circuit, the sensor interface module can only share some tens of μA or even lower. Fortunately, the sensor interface deals with low frequency and narrow bandwidth signals with medium dynamic range accuracy, which makes such low current consumption feasible.

There are several researches for portable ECG recording system. José Antonio

Gutiérrez Gnecchi proposed an Ambulatory Electrocardiogram Recorder [6], the ECGITM04. The 3-wire ECG monitoring device complies with several specifications: low-power consumption (battery operated), on-line graphics display, 7-days continuous data logger, patient electrical safety, minimal signal processing operations to facilitate the identification of cardiac arrhythmia patterns and a JTAG programming port so that the device can be updated without changing the data acquisition hardware. The system can maintain long time operation, but the size of this device is quite big. Patient may feel uncomfortable when wearing it.

Figure 2.3: José Antonio Gutiérrez Gnecchi's ECG system

I – Jane Wang proposed a wearable mobile electrocardiogram monitoring system [7] for long-term ECG monitoring. The wearable ECG acquisition device integrated with dry foam electrodes and the ECG acquisition module was designed for long-term ECG monitoring in daily life. Moreover, the ECG acquisition module is small-volume, wireless and low-power consumption. And based on SMS communication technology, patients can monitor their ECG anywhere in the globe if they are under the coverage

of GSM cellular network. The system is good in function but has a drawback that it has to use large capacity battery in order to maintain long time monitoring. In addition, dry foam electrodes are not weather proof.

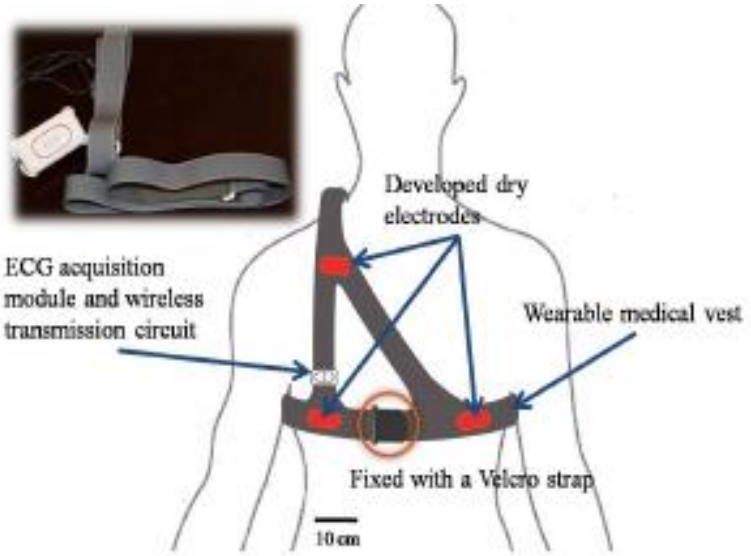


Figure 2.4: I – Jane Wang's device overview

2.2 Asynchronous Circuit

2.2.1 Introduction

The difficulty to find low power consumption is one of the crucial concerns for portable ECG monitoring system design. Otherwise, the battery cannot last very long time. The microcontroller, which is a significant source of power consumption for central control block, should have the desirable characteristic of low-power consumption. Hence, a technique for low power consumption design is needed.

Nowadays, most of the commercial digital designs are synchronous in nature. In

such circuits, there is usually a global clock signal which controls and synchronizes the data movement from one register to another. However, the power consumption of the clock tree constitutes a significant amount especially for low-power digital designs. Consequently, there is an increasing research interest in the field of asynchronous circuits over the years especially in the academic arena. Asynchronous circuits are fundamentally different from synchronous circuits in the way that there is no global clock signal present. Instead, asynchronous circuits make use of handshaking signals, which acts as local clocks that are not in phase and with varying period, to perform the controlling and synchronization of data movement as illustrated by Figure below. In this way, the registers in asynchronous circuits are only clocked where and when needed by the handshake signals.

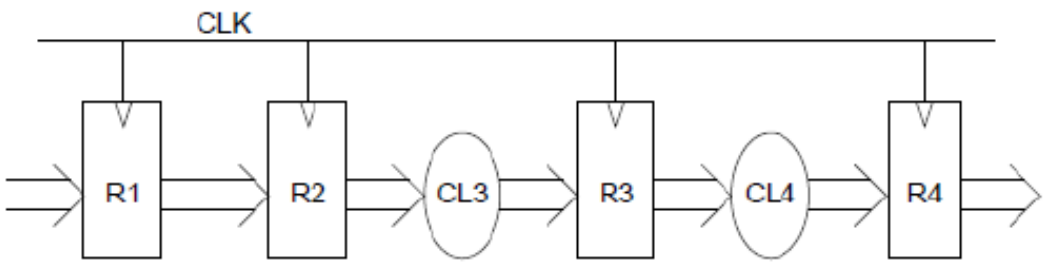


Figure 2.5: Synchronous pipeline stages controlled by clock signal [8]

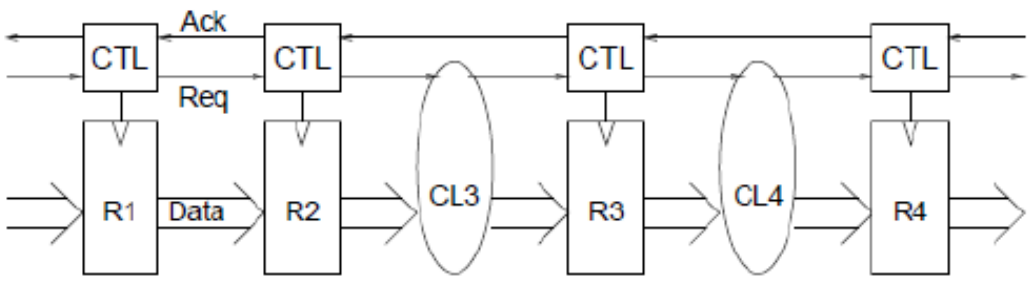


Figure 2.6: Asynchronous pipeline stages controlled by handshake signals

The main difference between synchronous circuits and asynchronous circuits lies in the data synchronization and communication method adopted. Compare to synchronous circuits, asynchronous circuits have several advantages. Firstly, asynchronous does not have clock skew problem, the absence of a global clock signal eliminates the clock skew problem faced in synchronous circuits. Secondly, asynchronous circuit power consumption for is lower than synchronous circuit. Absence of the clock tree in asynchronous circuits leads to practically zero stand-by power consumption when the circuits are idle. For some synchronous circuits with special sleep mode operation where the clock oscillator is turned off when the sleep mode is activated, they can also achieve practically zero stand variations in supply voltages and fabrication process. Timing assumption is based on matched delays for bundled data protocol, and for asynchronous circuits that adopt the dual-rail protocol, the insensitive or completely delay insensitive.

2.2.2 Asynchronous Handshake Protocols

In this project, the dual-rail four phase protocol is used to synthesize the asynchronous core of the 8051 microcontroller. A short brief is introduced here

For a 4-phase dual-rail protocol, there is always an empty state in-between two valid data. The handshake sequence is illustrated by Fig. 2.7 [8] and goes as follows: 1. The sender issues a valid data on the data bus, 2. the receiver sets the acknowledge line to logic 1 once it captures the valid data on the data bus, 3. the sender then issues

an empty data on the data bus after capturing a logic 1 in the acknowledge line, 4. the receiver accordingly lowers the acknowledge line upon detecting an empty data on the data bus, completing one handshake cycle.

.

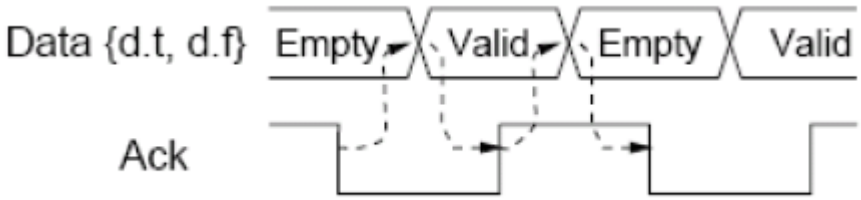


Figure 2.7: Handshake sequence of four-phase dual-rail data protocol

This protocol is very robust as it is insensitive to the delays involved in the wires connecting the two communicating parties. As it's so robust, voltage supply can be scaled down for the circuit design which use this protocol. Another reason to choose this protocol is that Balsa system can only support dual-rail four phase protocol in current version. In this work, asynchronous 8051 microcontroller adopts this protocol.

2.3 Design Tools

There are several design tools for implement Wearable ECG system and Asynchronous circuit.

2.3.1 Hardware Development Tool

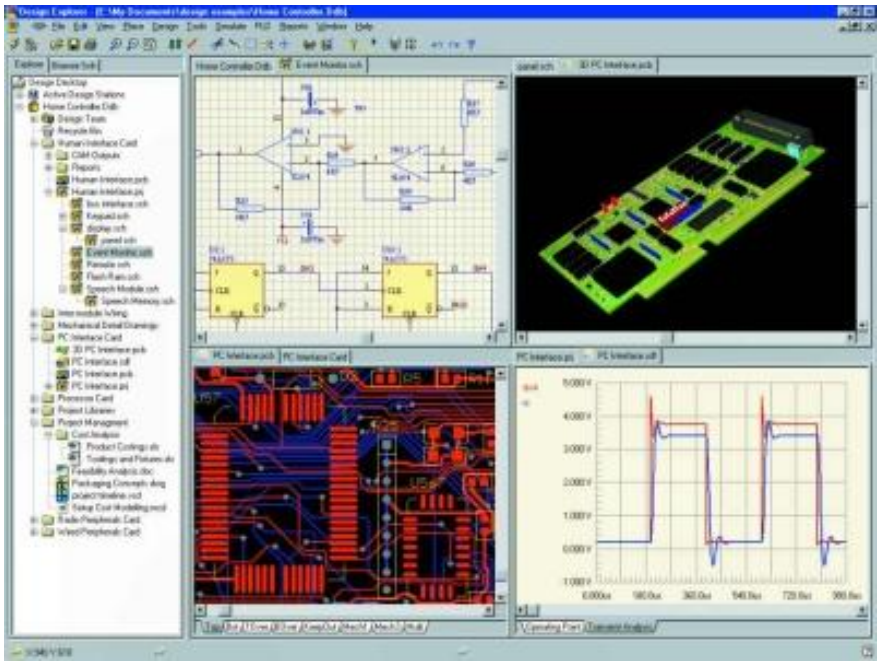


Figure 2.8: Altium Designer

This work's PCB design is using commonly used Altium Designer shown in Figure 2.8. Altium Designer is an EDA software package for printed circuit board, circuit and layout design

2.3.2 Firmware Development Tool

In order to design the firmware of Energy efficient wearable ECG system, C programming development tool is needed. MPLAB Integrated Development Environment (IDE) is a free and officially supported development environment application, which could integrate with many third party compilers and fully support ICD2 device. It can highlight the codes and organize different files in one project. With the help of In-Circuit-Debugger 2 (ICD2), MPLAB can trace the code line by line. Here, MPLAB IDE V8.60 is used for developing firmware.

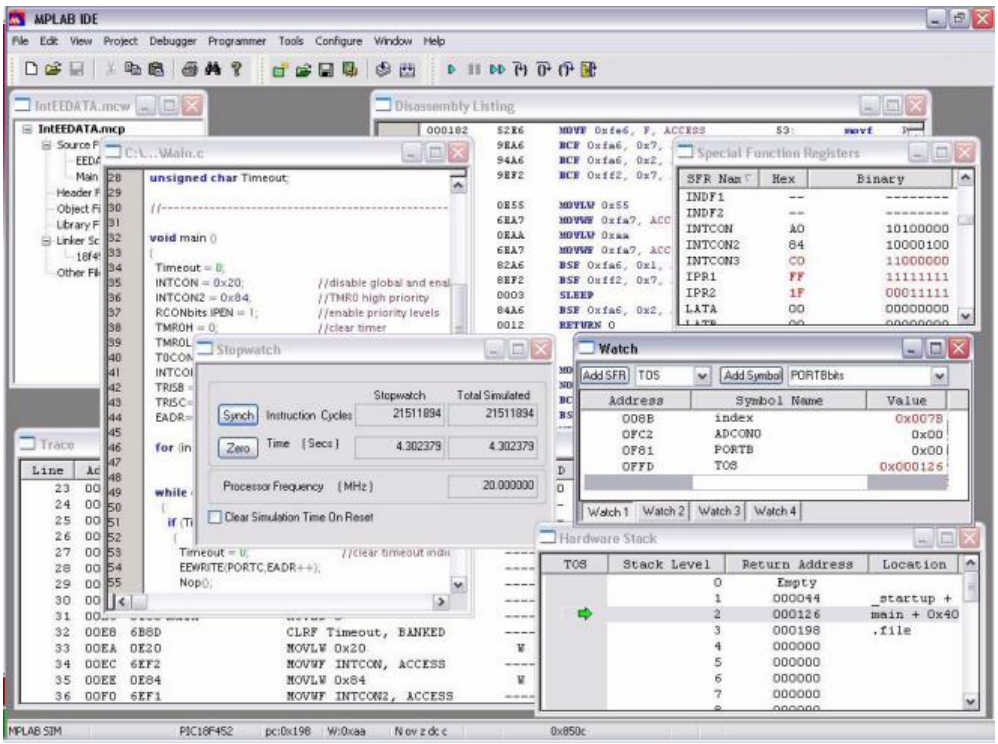


Figure 2.9: MPLAB IDE

2.3.3 Balsa for Asynchronous Circuit design

Balsa [9] is software for Asynchronous Circuit design. It provides a fully automatic approach for synthesizing asynchronous circuits through describing the asynchronous circuits using a hardware description language – Balsa language. Asynchronous design is first described in the Balsa language. Through a compilation, the Balsa description is transformed into the intermediate breeze description which is a netlist composed of various handshake components. Behavioral simulation can be formed on this handshake component (HC) netlist using the Balsa behavioral simulation system for initial verification. After this, it will convert to a HDL file such as verilog and VHDL for

Synopsys or Cadence to use.

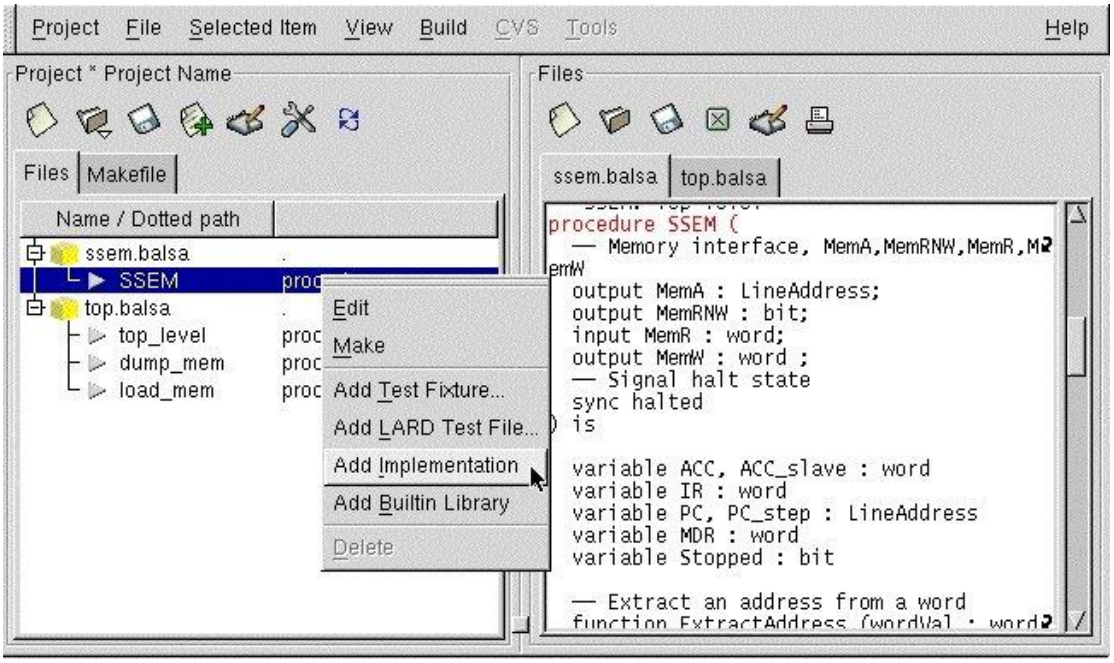


Figure 2.10: Balsa

Chapter 3

Wireless ECG Plaster

3.1 System Overview

The design objective for Wireless ECG Plaster is to conduct a real-time monitoring of ECG in cardiac patients. This device continually records patient's single-lead ECG signal and wirelessly stream it to a remote station for monitoring and analysis, using a ZigBee transceiver. The proposed device extremely light weight at 25 grams and easy to wear, and therefore is comfortable.

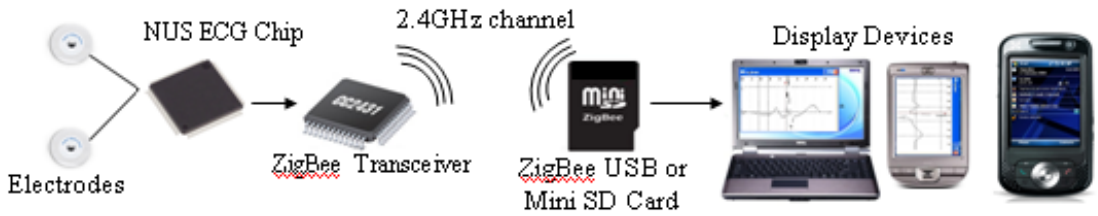


Figure 3.1 System Overview

The overall system includes two parts: (1) a wireless ECG acquisition plaster, and (2) a personal gateway (or remote station) as shown in Fig 3.2. The ECG plaster contains a custom designed ECG front-end chip, a microcontroller, and a ZigBee transceiver. The personal gateway can be either a mobile phone or a PC with a USB

ZigBee interface. The plaster records the ECG and wirelessly transfers the data to a remote data center through the personal gateway.

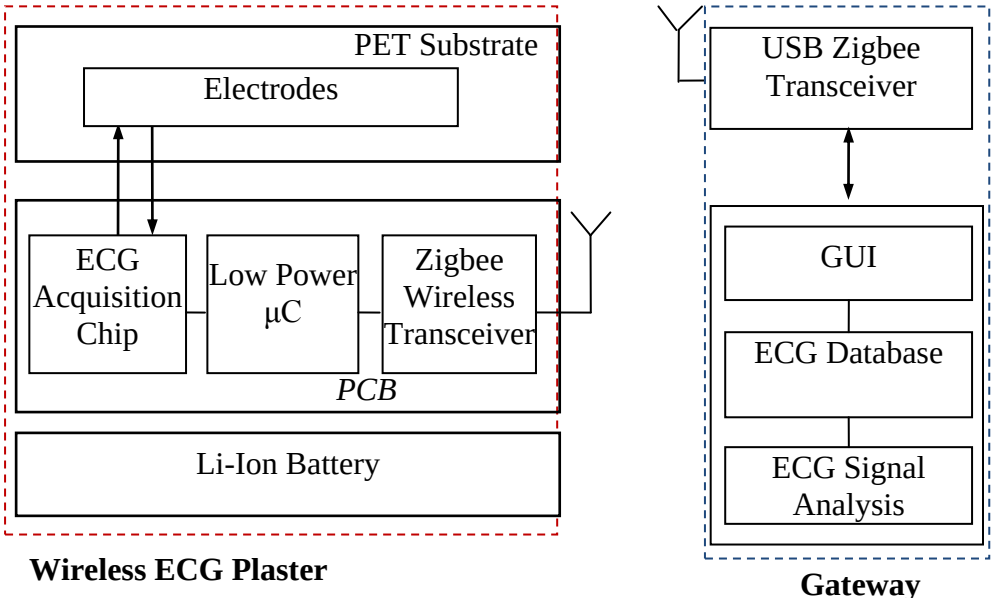


Figure 3.2 System Architecture

The ECG acquisition chip is designed for low power. The details will be presented in next part. For wireless communication, ZigBee (TI CC2420) is selected as it offers sufficient data rate at reasonable power consumption. The MCU (TI MSP430) is used for ZigBee baseband and for ECG data management. The plaster was designed with user comfort and ease of use in mind. Hence, it does not affect the daily activities of users. In addition, the plaster is sealed with splash and water-proof material, so the patient can take shower with the plaster.

3.2 Hardware

In order to design Energy efficient wearable ECG system, the power consumption of each hardware component on the PCB must remain low. The Table 3.1 shows the major component of Wireless ECG Plaster.

Table 3.1: Hardware components

No.	Component	Function
1	BMDAV8	ECG Acquisition chip
2	MSP430F2254	Microcontroller
3	TI CC2420	ZigBee wireless transceiver
4	TPS73615-EP	Regulator
5	Hi - Charge Li - ion battery	3.7V 650mAH battery

3.2.1 ECG Acquisition chip BMDAV8

First of all, a NUS ECG Acquisition chip BMDAV8 is selected for this project. The BMDAV8 is a low-power biological data acquisition device that is targeting pervasive healthcare and medical apparatus market. Optimized for battery-powered applications, its core circuit consumes approximately 30 μA of current with 3-V supply, and promises over 10 bits of effective resolution with up to 25 kS/s of sampling. The detail of this chip is illustrated in Figure 3.3.

For a low-power weak-signal pickup device, one of the most essential links along the acquisition chain is its analog processing frontend and analog-to-digital interface. The required low noise, low distortion analog capabilities always conflict with the limited power budget. Unfortunately such situation does not scale down with process technology as well as in digital domain, and in fact usually gets worse with more

advanced process nodes. In our proposed ECG plaster, we use a proprietary biomedical data acquisition frontend chip that employs and extends the solutions we demonstrated in [10] [11].

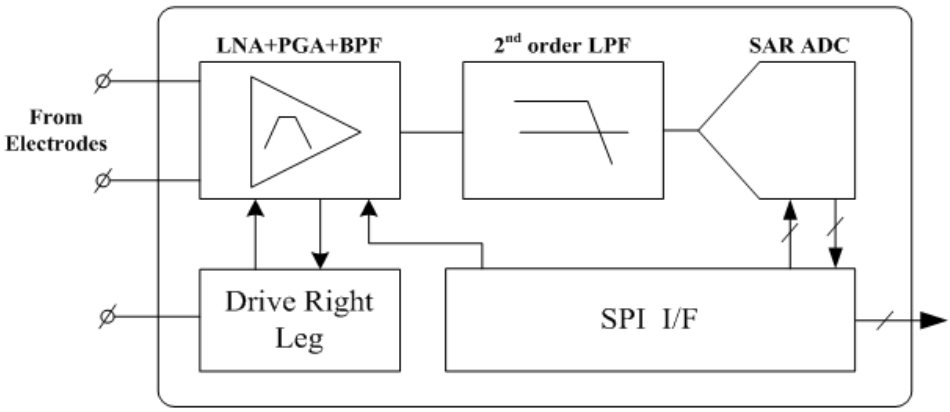


Figure 3.3: Architecture of Proposed ECG Acquisition Chip

As shown in the block diagram in Figure 3.3, the chip houses a fully featured bio-signal acquisition frontend, with all necessary tuning functions to cater for different input conditions. The front-end amplifier has on-chip high-impedance DC-blocking inputs that can be directly applied to ECG electrodes. The amplification stage consists of a low noise front-end amplifier with band-pass function and a programmable gain amplifier (PGA) employing the flip-over-capacitor technique [10], as shown in Figure3.4. Both op-amps are biased in subthreshold mode to ensure optimal noise efficiency against power. During startup or after an input interruption event such as electrode falloff, a reset signal is asserted to eliminate the large time constant associated with the high-pass filter, such that the preamplifier can quickly resume operation. A series of secondary low-pass filters then provides further suppression to the out-of-band residues such that lower sampling frequency (in this case three times

of signal bandwidth for over 20-dB attenuation) that favors lower wireless bit rates can be used. Following the analog processing modules, a 12-bit charge redistribution SAR ADC quantizes the conditioned ECG signal based on the sampling speed set by the microcontroller, and encodes the data into 16-bit SPI frames.

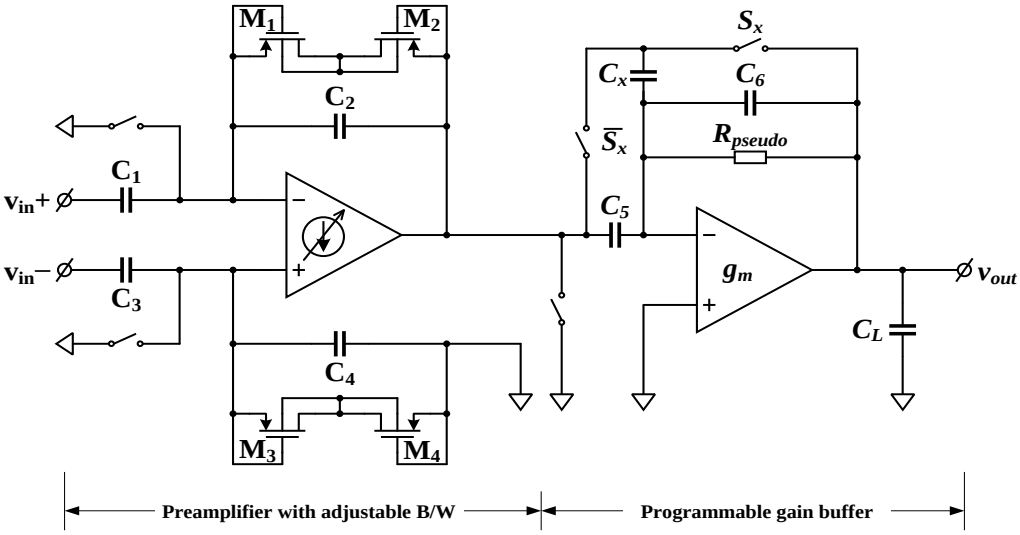


Figure 3.4 Circuits for the ECG frond-end

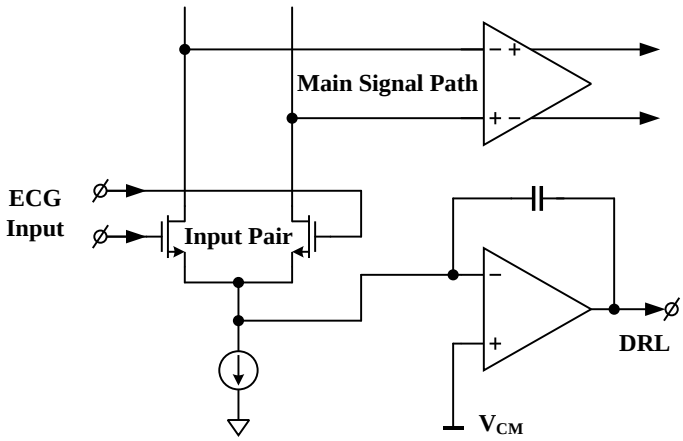


Figure 3.5: Concept of low power DRL circuit with direct common-mode extraction

Alongside the main signal path, supporting circuits help to ensure the signal integrity, among which two micro-Watt right-leg drivers (DRL) prove to be most effective in counteracting common-mode interferences (namely power line interference) and excessive electrode contact resistance. Here DRL1 employs a novel sensing structure, where the common-mode interferences are directly extracted from the main signal path without the need of dedicated sensing circuitry, facilitating further power saving. The concept is illustrated in Figure 3.5.

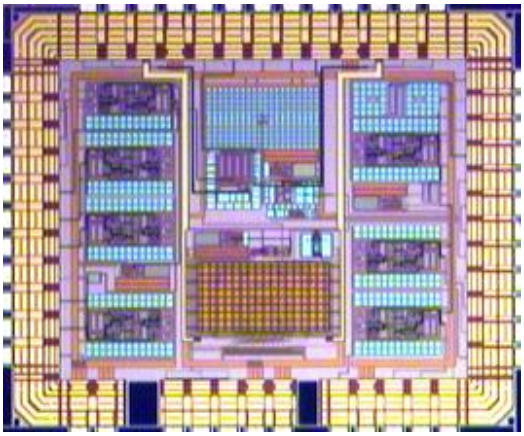


Figure3.6 Chip micro photo

Table 3.2 Performance Summary

Supply Voltage	1.8 ~ 3.6 V
Technology	0.35μm
Low-pass Frequency	50 Hz ~ 8 kHz tunable
Gain	47 ~ 64 dB tunable
Input-referred Noise	3 μV_{rms} (0.05 ~ 500 Hz)
THD @ FS Output	< 1%
Sampling Freq	up to 25 kS/s
ADC ENOB	> 10.3
Interface	SPI slave
Current @ 3 V, ECG mode with DRL	18 μA

With all the innovative power saving measures implemented, the entire chip consumes less than 18 μW and 50 μW when operates at 1.8 V under ECG mode with DRL turned off and on, respectively. Some of the key specifications are summarized in Table 3.2. The chip die photo is shown in Figure 3.6.

3.2.2 Microcontroller

The MSP430F2254 is a commonly used mixed signal microcontroller with two built-in 16-bit timers, a universal serial communication interface, 10-bit A/D converter with integrated reference and data transfer controller (DTC), two general-purpose operational amplifiers in the MSP430x22x4 devices, and 32 I/O pins.

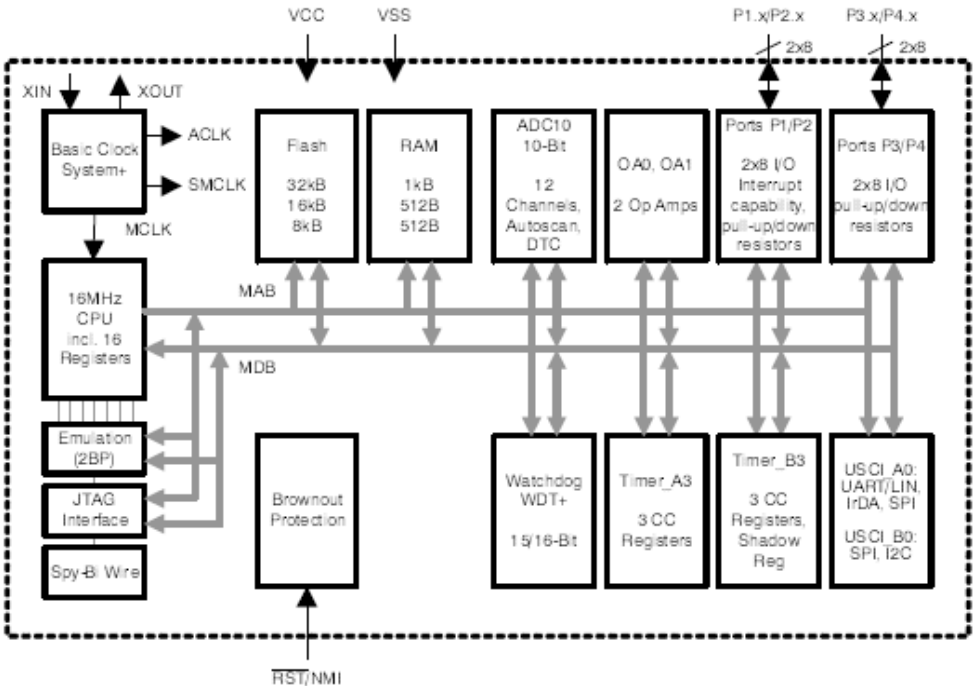


Figure 3.7 Microcontroller MSP430F2254 block diagram

The major concern to select MSP430 (Figure 3.7) as central control unit for Wireless ECG Plaster is below

- The MSP430F2254 3.3V ultra low power microcontroller consists of several devices featuring different sets of peripherals targeted for various applications.
- 0.7 μA standby current to save power during idle
- UART & SPI interface for faster data transmit

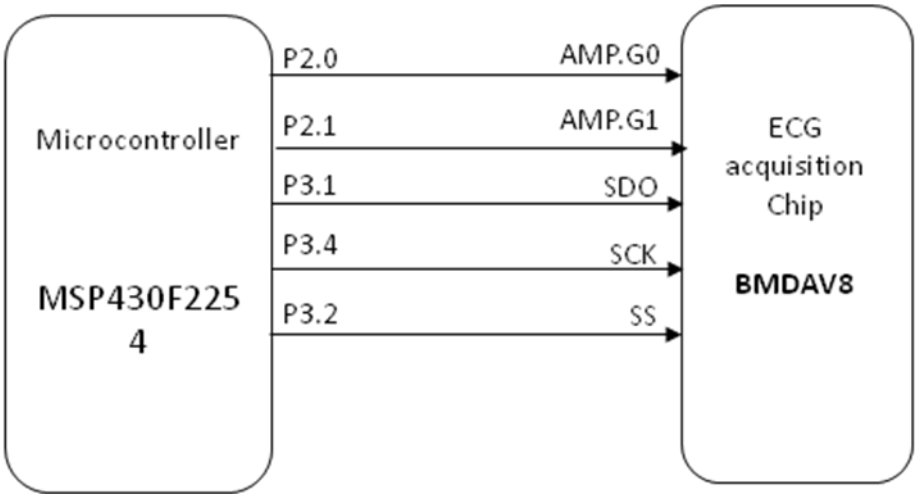


Figure3.8 Configuration for microcontroller and BMDAV8

The pin configuration between MSP430 microcontroller and BMDAV8 ECG acquisition chip is shown in Figure 3.8. MSP430 can control the ECG signal gain of BMDAV8 by 2 outputs P2.0 and P2.1. The outputs P3.0 to P3.4 are used to collect ECG signal information though SPI interface.

3.2.3 Zigbee RF transceiver

The Zigbee RF Modules were used for wireless communication between gateway and Plaster. It has several features below for us to select this component.

Figure3.9 CC240 Zigbee RF transceiver

- Key feature is that CC2420 is easy to use as it will handle the difficult part like hand shaking by itself. It is engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks.

- The modules operate within the ISM 2.4 GHz frequency band. Its transmitting and receiving current is around 50mA at 3.3V and its indoor/urban range can up to 30m.

3.2.4 Electrode and PET substrate

Last but not least, ECG monitoring system needs medical contact to collect ECG signal. Most market ECG devices use traditional ECG lead contacts, which were not

designed with wearability in mind, and have multiple wires hanging around the body. In this work, an ultra-wide sensory mesh based electrode structure is specially designed for the proposed device. The electrode is made using a highly conductive silver ink built on to PET substrate.

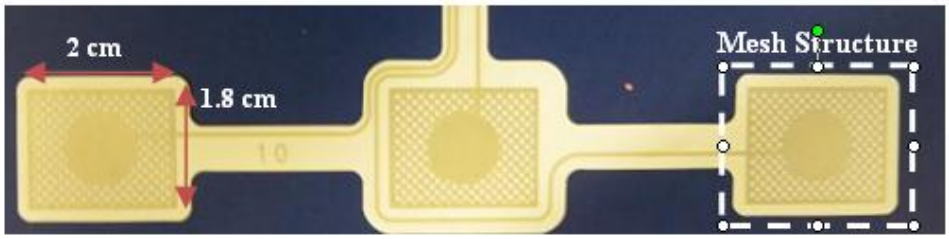


Figure3.10 Mash structure Electrode

The plaster comprises of materials from the latest stick-to-skin technologies from 3M. These medical-grade materials have been proven to be biocompatible, hypoallergenic, breathable, and water-proof for over 7 days, even during adhesion to human skin.

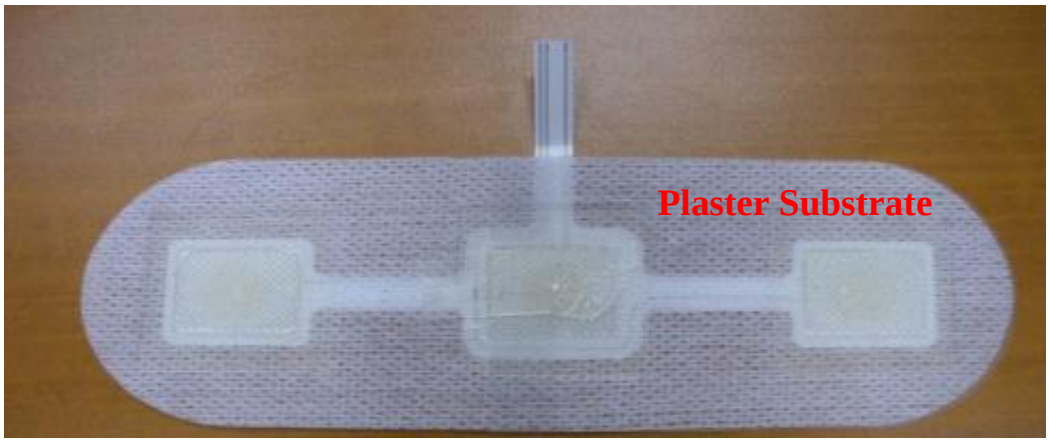


Figure3.11 Plaster substrate

After integrating all the selected low power components in the above, A PCB

board is developed using Altium designer.

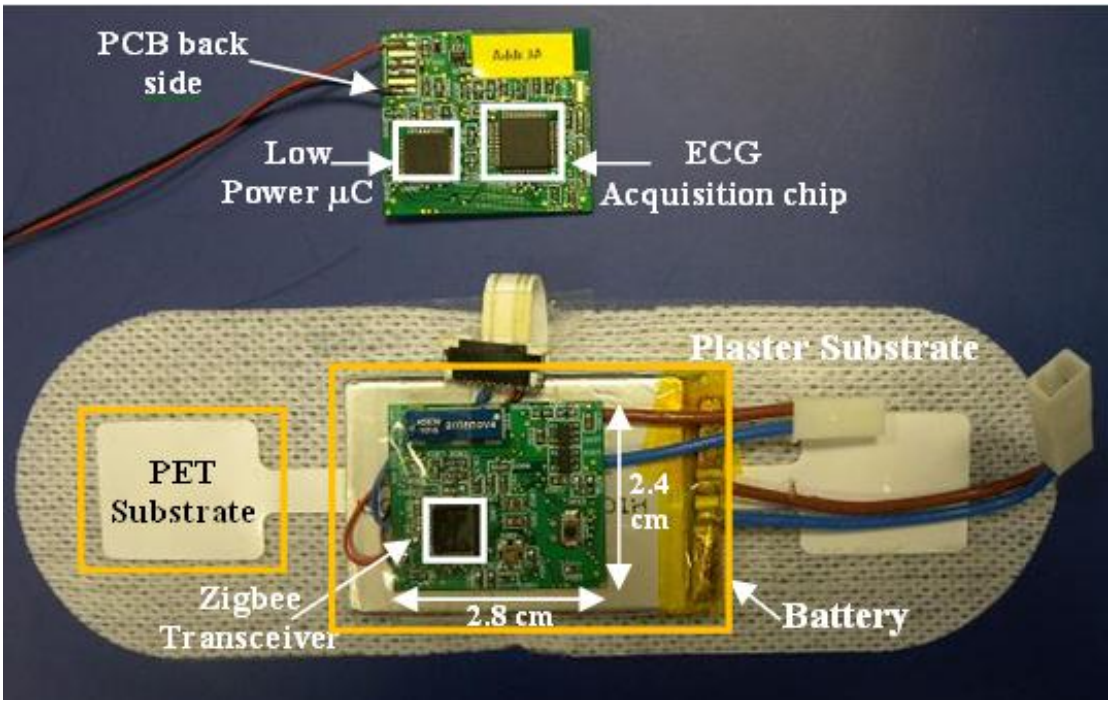


Figure 3.12 Wireless ECG Plaster Top view

In short, a prototype of wireless ECG plaster is shown in Figure 3.12. It consists of: (1) a specially designed skin electrode plaster for acquiring the ECG; (2) a miniature printed circuit board (2.8cm x 2.4cm) with our proprietary ECG front end chip; (3) and a high density 650mAH rechargeable Lithium Ion battery. To minimize power consumption, the data is buffered using MCU internal memory before sending to the gateway wirelessly. The maximum range of ZigBee transmission is about 15 meters in the room. The operational time is around 26 hours for each charge.

3.3 Firmware

The firmware of wireless ECG plaster is written in C code. It performs the following tasks: ECG front-end and microprocessor initialization, managing ECG data buffering, and scheduling the ZigBee transceiver. A brief introduction of firmware is shown in a flow chart below

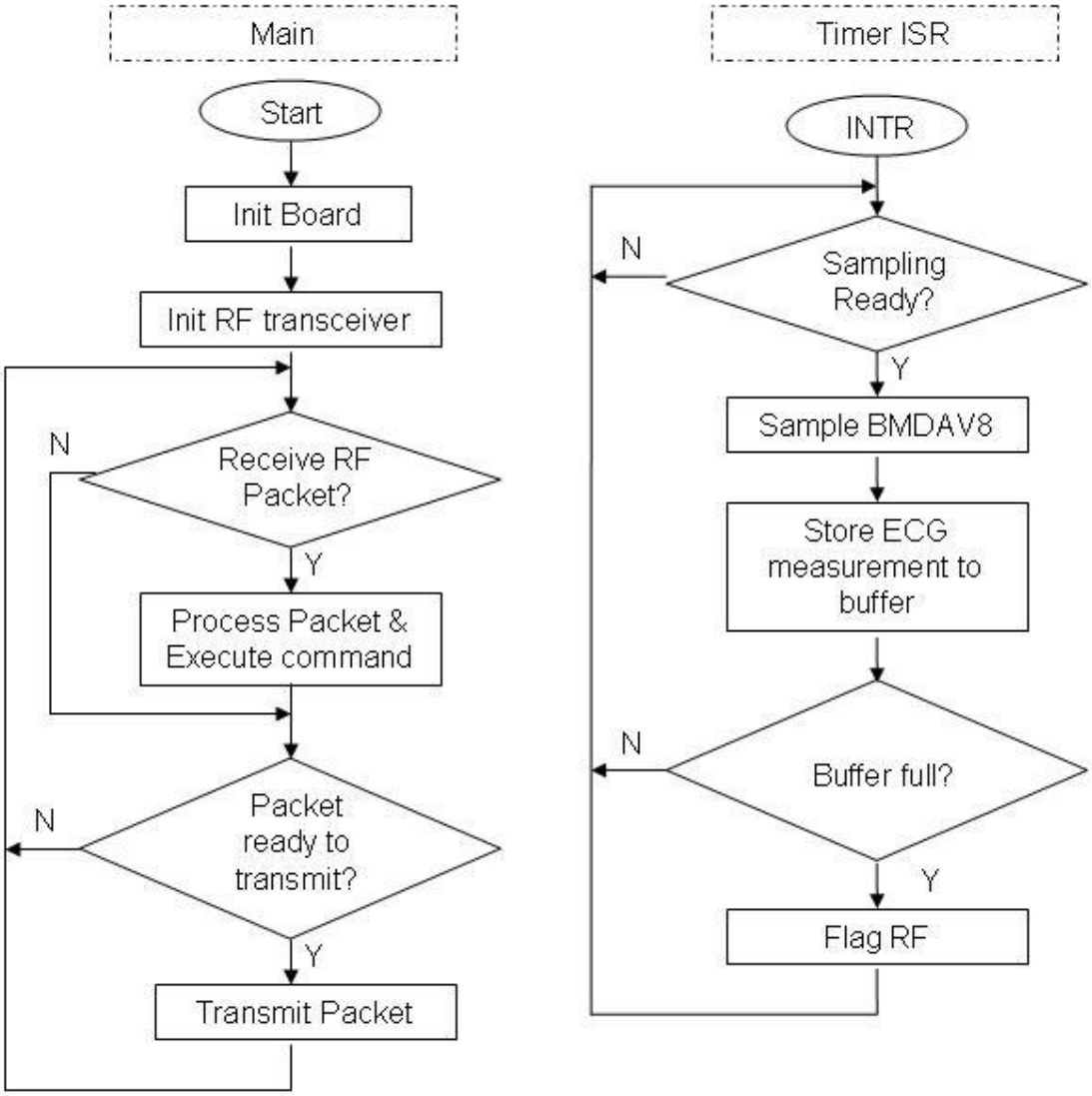


Figure3.13 System Firmware Flow Chart

In Wireless ECG Plaster, PC is the Gateway (master device) to send control signal to control ECG plaster's operation all the time. However, the firmware on the ECG

plaster handles the ECG data transmission. After initializing ECG acquisition chip and ZigBee transceiver, the firmware will keep listening to the RF channel, for any changes in the control settings, issued by the PC application. Any such modifications are immediately updated, by making necessary changes in the register settings of the corresponding chips on the plaster. After that the ECG signal acquisition starts and the sampled data is temporarily buffered locally. During this time, the ZigBee transceiver is put in sleep mode in order to save power. Once the amount of data buffered locally becomes large enough to send a ZigBee packet, an interrupt will be raised, to switch on the chip and initiate a transmission. ZigBee chip consumes the most power in our device, and this buffering mechanism helps to reduce the power consumption. Also the payload size in each packet is selected (as 64bytes) as a trade-off between "header overhead" and "collision probability", in order to reduce the overall system power.

3.4 Graphical User Interface

Figure 3.14 shows a sample application of GUI interface on PC for receiving and displaying ECG data. This user interface is implemented by LabVIEW. User can monitor real time ECG signal though this GUI. It receives the ECG data package from wireless ECG plaster by using a USB ZigBee transceiver. In order to avoid signal interference from other wireless devices, the GUI interface can switch between 15 wireless channels. This also allows up to 15 patients to be monitored simultaneously. In addition, there are several function buttons in the GUI interface for changing

parameters of the plaster, such as sample rate, overall gain and low-pass filter. These buttons are located at the right side and bottom part of the GUI.

Figure 3.14 GUI interface for PC.

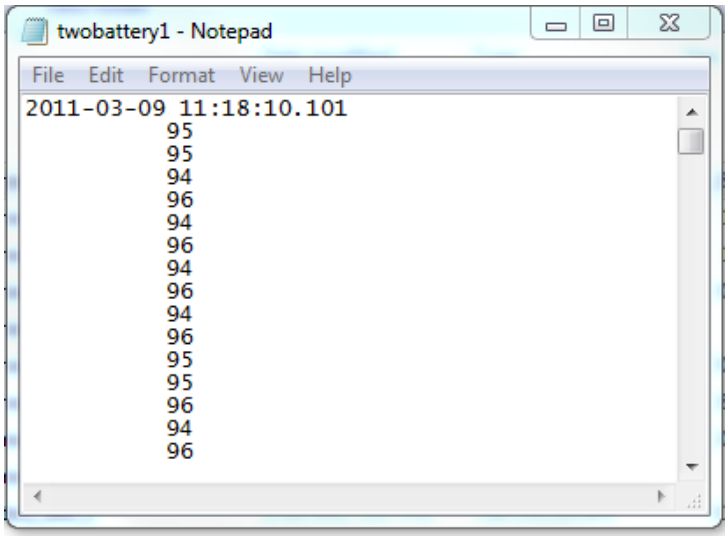


Figure 3.15 ECG data file saved from GUI

GUI can save the patient's ECG data into a text format file. For example, In the

Figure 3.15 the sample rate is set to 100 which it means there will be one real-time information (Date, time) and 100 ECG data information saved each second. All these information will be saved second by second to form a complete ECG information record. With the help of detailed ECG information saved by wireless ECG Plaster, doctors can easily diagnosis patient heart disease.

3.5 Design Verification

The objective for Wireless ECG Plaster is to push it into the market. In Singapore, the standard for ECG monitoring prototype becomes a commercial product is quite high. A lot of clinical trial data has to be taken in order to prove the system working accurately and harmless. To verify the accuracy of the system, two clinical trials were conducted by doctors at two hospitals which are National University Hospital and Singapore General Hospital. In these two trials, the radio frequency channel is centered at 2405MHz. The ECG sampling rate is selected as 100Hz. The input signal gain is 47dB for first trial and 56dB for second trial. An embedded low pass filter is used in the trial to remove 50Hz noise.

3.5.1 System Accuracy

The first trial was to verify the accuracy of the device. The First clinical trial was at National University Hospital on February 25th 2011. The objective of this trial was to test system performance when subject carry out normal daily activities. This subject was a healthy male candidate. A wireless ECG plaster and a commercial ECG monitoring product were used to monitor the subject's health status at the same time.

The location of the two devices is shown in Figure 3.16. The lead configuration for ECG plaster is pseudo-aVL (approximate 2/3 of aVL). A portable PC with our software application was carried by the patient for ECG recording. At the end, two sets of continuous one-hour ECG recording were been collected from wireless plaster and the reference holter.

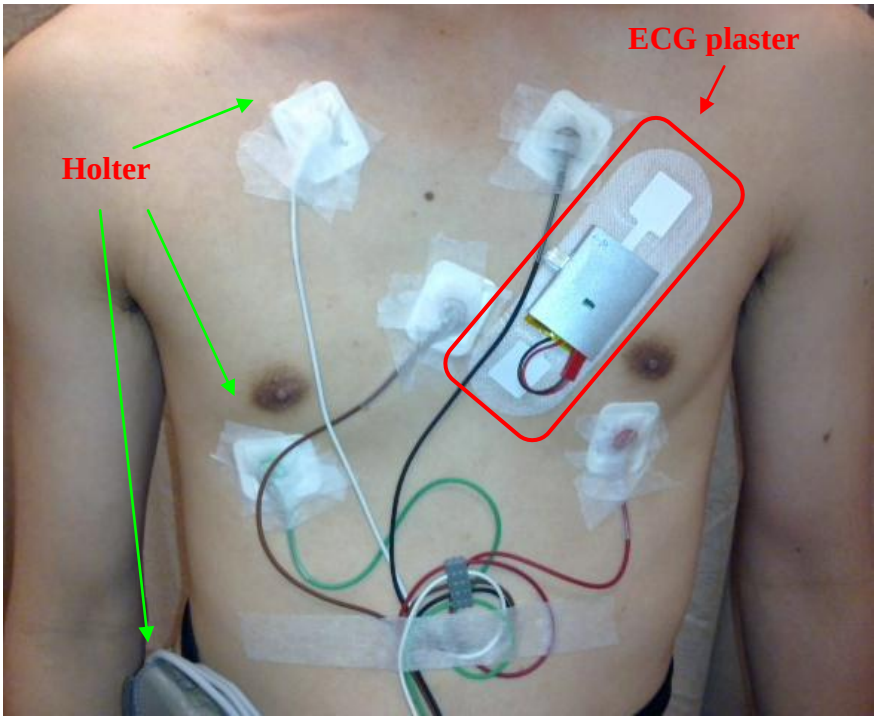


Figure 3.16: The positions of the wireless ECG plaster and Holter

Figure 3.17 shows two ECG records from the reference commercial Holter (Channels 1 and 2) and the proposed wireless plaster, respectively. We use several methods to verify the quality of ECG obtained using our device.

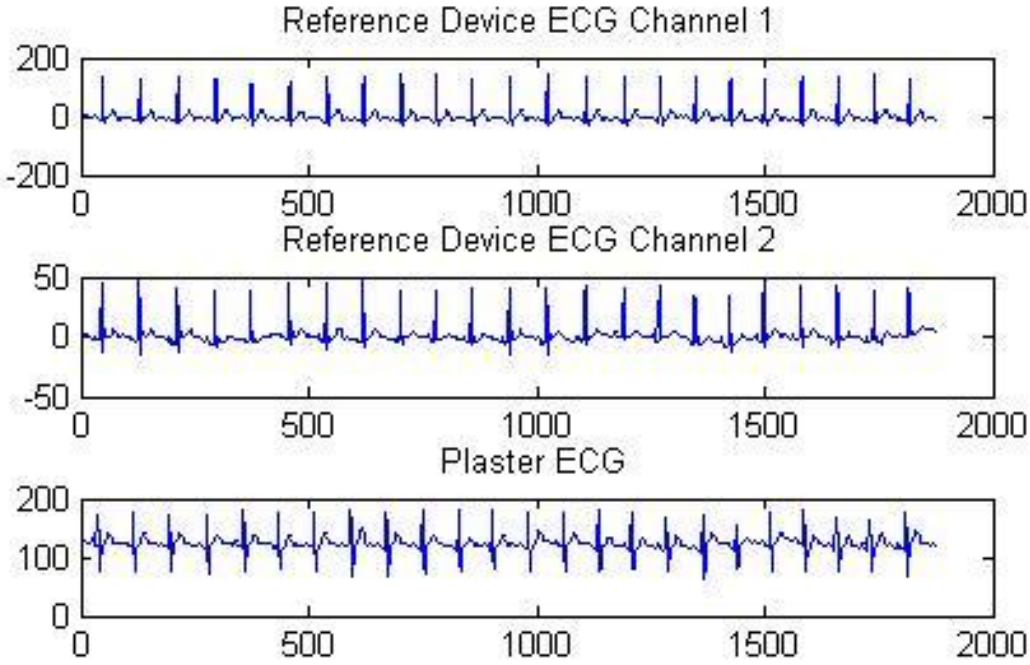


Figure 3.17: ECG Signal: Plaster Device Vs Reference Holter Monitor

Method 1: Average Heart Rate and QRS peaks.

The data collected from the proposed device and reference Holter are analyzed using popular QRS detection algorithms [12] in Matlab. From the simulation, it was observed that the number of QRS peaks detected by the algorithm, in a one hour ECG data set obtained from a patient using both devices, varies by only 0.4%. A few QRS peaks were missing in our device due to the error caused in Zigbee wireless transmission. The average Heart rate estimated using both data sets for the same patient is 99.05bpm and 99.48bpm, respectively.

Method 2: RR Interval

In order to establish the equality of ECG obtained from both devices, we compute the RR interval for every beat in the ECG signal. The average differences in RR interval obtained using both devices are found to be less than 1% of the reference device. The histograms showing the RR interval for both data sets are shown in Figure 3.18.

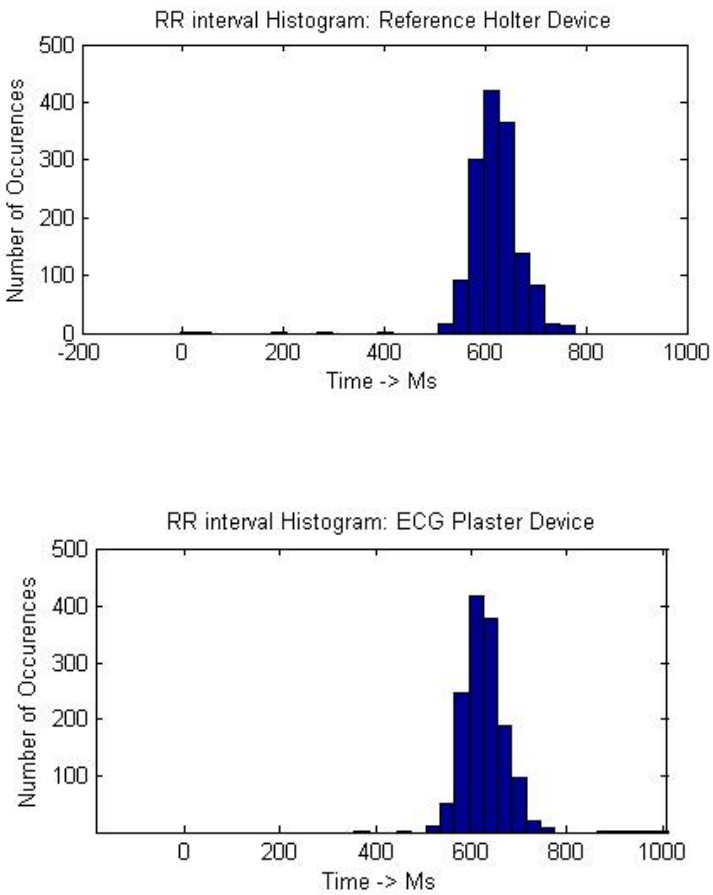


Figure3.18: RR Interval histograms: ECG plaster Vs Reference Device

3.5.2 System Reliability

The second trial was to verify the reliability, stability of the device and wireless link. The second clinical trial was at Singapore General Hospital on March 23rd 2011.

A healthy male adult subject was monitored by wireless ECG Plaster for more than 40 hours. In order to extend the operational hour, two 650mAh batteries were combined in this trial. The subject was isolated in a special ward designed for clinical trial. The plaster was pasted on the position V2 to V4. A laptop with an USB ZigBee transceiver is used to collect the data.

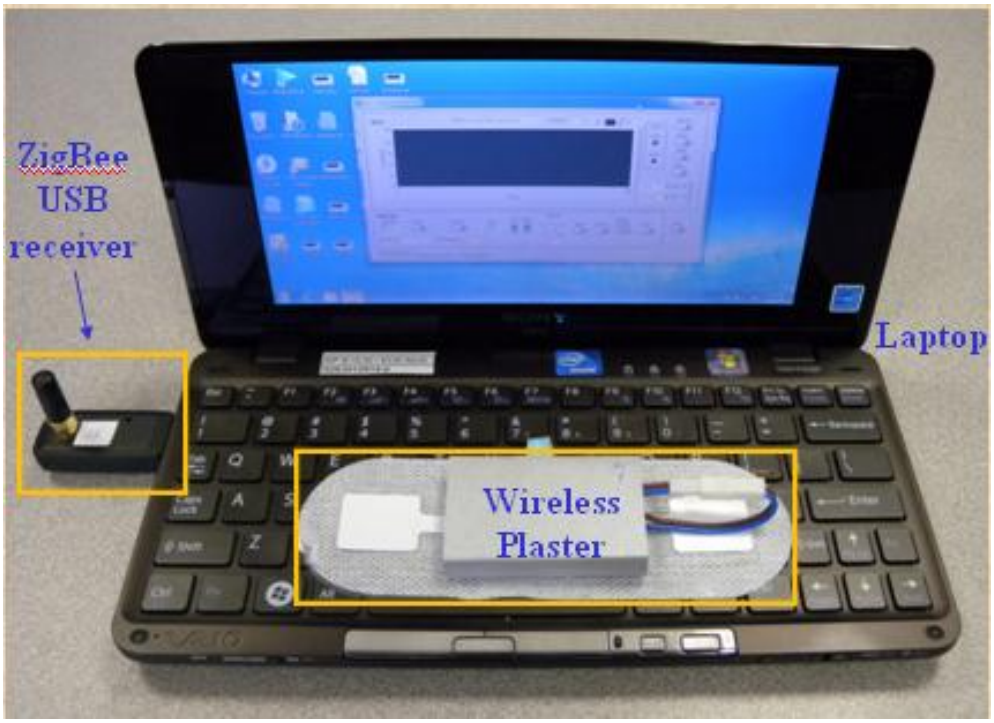


Figure 3.19: SGH Clinical Trial set

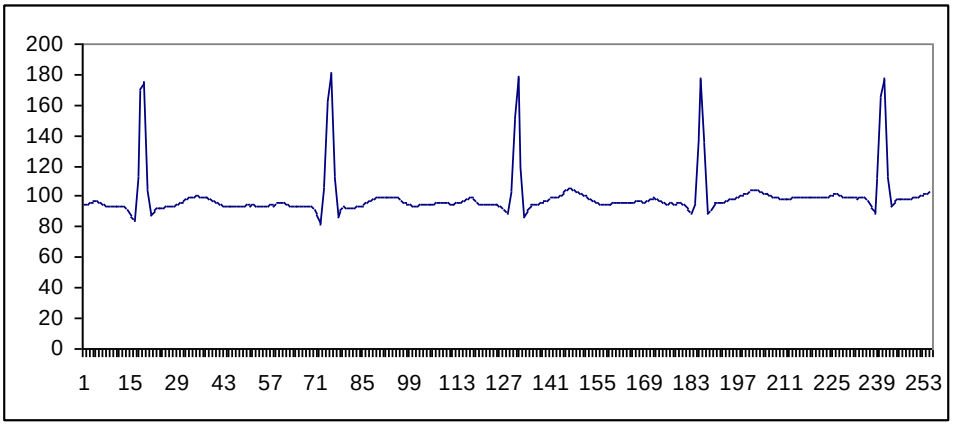


Figure 3.20: Subject 2nd day morning Record

The data recorded for 40 hours in the second trial shows the proposed wireless ECG plaster works reliably. The plaster can record a clear ECG data as shown in Figure 3.20 in the second day. The normal daily activities do not have a significant impact on the plaster. As a result, the PC in the room successfully collected continuous 40 hours ECG data. The reliability of wireless transmission and stability during long time operation has been verified.

In short, the Wireless ECG plaster has good system accuracy and reliability. The device characterization is shown in Table 3.3.

- System accuracy
  - Average heart rate and QRS peaks: nearly same
  - RR interval difference: less than 1%
- Reliability and stability:
  - continuous 40 hours monitoring success,
  - Rarely ECG signal drop.
  - Wireless transmission range within 15 meters

Table 3.3: Wireless ECG Plaster summary

Property	Wireless ECG Plaster
PCB Size	2.8cm x 2.4cm
Plaster	Yes
Wireless operating range	15 meters Maximum
Gain	47 - 64 dB
Sample rate	up to 25K S/s
Battery supply	3.7V 650 mAH Li-ion battery
Current Consumption	25 mA
Continuous running time	26 hours
RR interval difference	< 1%
Accuracy	> 99 %

In conclusion, Wearable ECG Plaster is designed for real-time cardiac health monitoring. The proposed device is wearable, light weight, comfortable and can wirelessly transfer the patient's ECG signal to a remote monitoring station, where it can be analyzed in detail. The device has a battery life of around 26 hours while in continuous operation. However, this system can't satisfies for the long time recording purpose. Hence, a new solution is described in next chapter.

Chapter 4

Long Playing Cardio Recorder

4.1 Overview of LPCR system

In recent times, Wearable ECG record products like Holter are popular for Doctors monitoring patient when they are at home. However, due to the limited size of portable ECG devices, they cannot save very long time data. A short term record cannot completely represent the patient health status. Doctors prefer longer time data result for more accurate analysis. Most ECG recoding commercial products in the market can last for 24 hours. Under some situation, patient's heart may work as normal in a certain day. Hence, these ECG recording products cannot save the key value for heart problem. It's difficult for doctor to diagnosis what kind of heart disease patients may have, the treatment will be delayed. On the other hand, a device can make long time record to solve this problem.

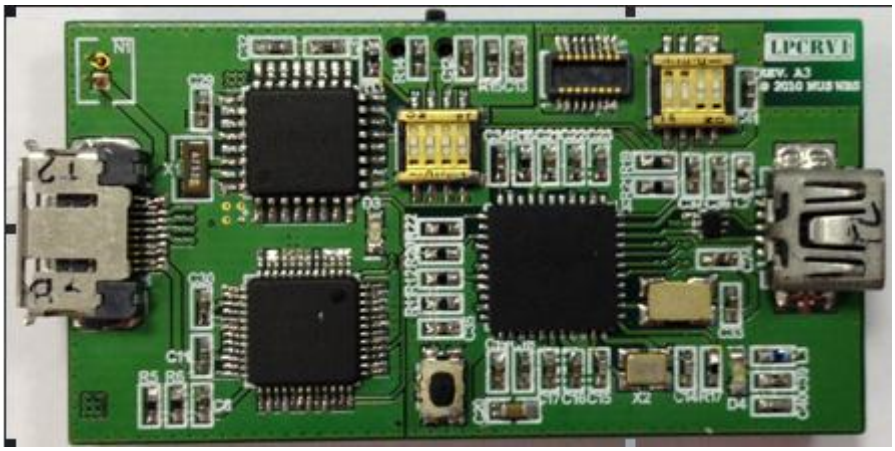


Figure 4.1: Long Playing Cardio Recorder (LPCR) Overview

As shown in Figure 4.1, Long Playing Cardio Recorder (LPCR) is specially developed firmware could make the device continuously recording ECG data for more than twenty days with a fully charged Li-ion battery (650mA). In addition, the benefits of light weight, smaller size and long-period operating time are still the primary objective for hardware design. Hence, LPCR is a good solution for doctors to track patients‟ heart activities”. LPCRV1 is the first version of LPCR system. The difference between LPCR and Wearable ECG Plaster is that LPCR uses 3 commercial ECG wires shown in Figure 4.2 to collect data instead of wearable ECG plaster. LPCR system has two kinds of data transmission methods from its PCB to computer. One is for real time monitoring by using Bluetooth communication, and the other is to transmit big size long time ECG data by using USB port.

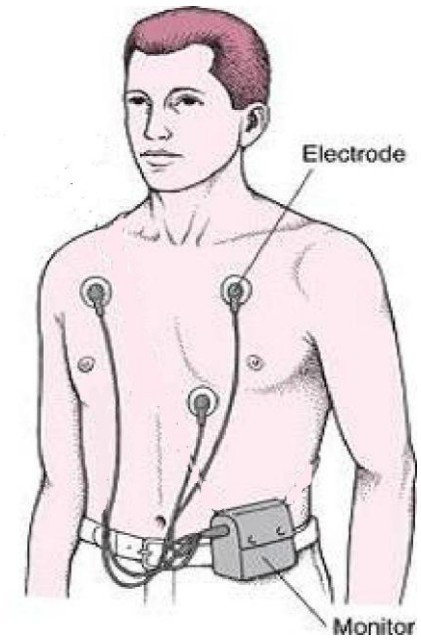


Figure 4.2: LPCR ECG data collecting method

Figure 4.3 shows the block diagram of LPCR hardware. There are several major components integrated in this system. A microcontroller PIC18F46J50 as a central control unit, an ECG acquisition device BMDAV7 for ECG data collection, a 16Gbit NAND Flash chip MT29F16G08DAAWP to keep a long time results. These three chips are working together as local data processing part. For communication part, a Bluetooth module Bluegiga WT12 is selected for wireless communication and a small USB connector for data transmission. The power supply is coming from a 3.7V 650mAh Hi-Charge Li-ion battery.

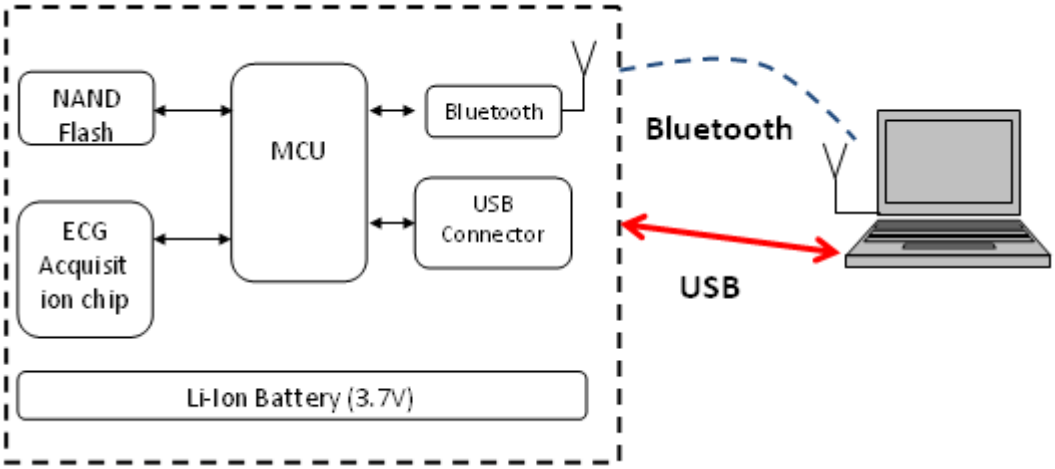


Figure 4.3: Block Diagram of LPCR system

Lower power consumption is the most important goal for this device. This work will focus on hardware design concern for microcontroller and Flash chip, and developing firmware to make local data processing part work very well. In addition, system reliability and battery life testing will be illustrated at the end of this chapter. The firmware of communication part will not be explained in detail because its power consumption is controlled by the computer.

4.2 Hardware

A 5.0cm x 2.7 cm prototype PCB is built for LPCR systems operation (Shown in Appendix 1). There are around 15 major hardware components in LPCR system except capacitors, resistors and diodes as shown in Table 4.1. Microcontroller, Flash Memory, ECG Acquisition chip and blue tooth Module are explained in detail in section below.

Table 4.1: LPCR system Hardware major components

#	Component	Function	Value
1	PIC18F46J50	Microcontroller	
2	MT29FXG16XXX	Flash Memory	8Gb/16Gb
3	BMDAV7 (QRS)	ECG Acquisition chip	
4	Bluetooth Module	Wireless transition	
5	TPS61132	Converter	3.3V/1.5V
6	LED	Indicate ECG signal	
7	Crystal		32.768KHz
			16MHz
8	DIP Switch	Operatiing mode chage	8-way
9	Slide Switch	Power	
10	Push Button	Reset button	
11	USB Connector	Communicate to computer	5-way
12	Input Connector	ECG signal input port	10-way
13	Power Connector		2-way
14	BMDAV8	DRL support	
15	Service port	Programming socket for MCU	
16	Li-on Battery		3.7V 650 mAh

4.2.1 Microcontroller

The microcontroller is a crucial part in this work. PIC18F46J50 [13] is a new line of low-voltage Universal Serial Bus (USB) microcontrollers with the main traditional advantage of all PIC18 microcontrollers, namely, high computational performance and a rich feature set at an extremely competitive price point. Figure 4.4 Block diagram of PIC18F46J50 shows its features.

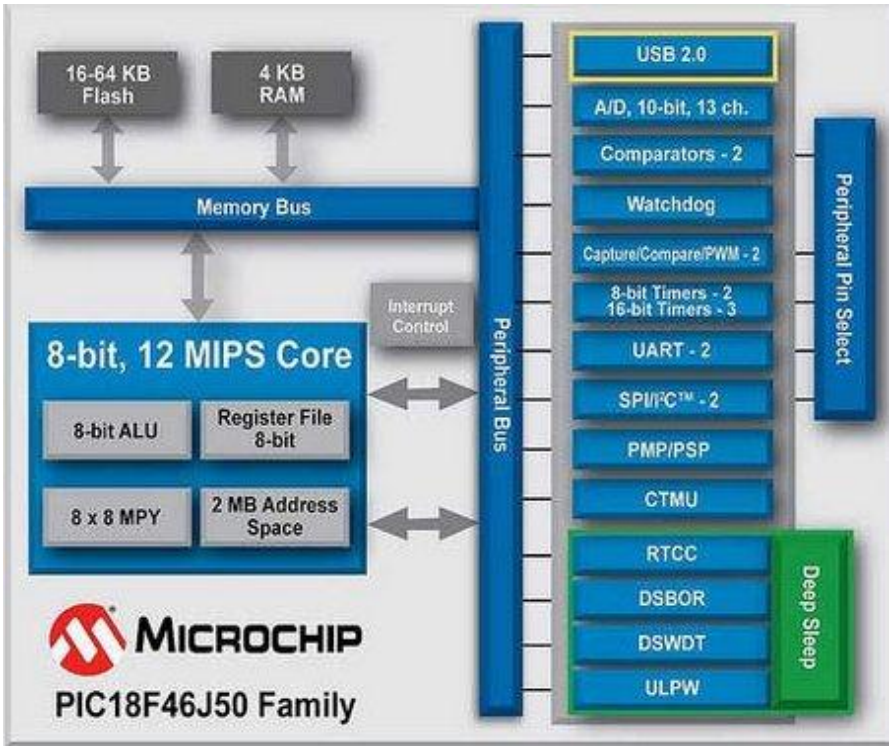


Figure 4.4: PIC18F46J50 block diagram

There are 3 concerns to choose PIC18F46J50 for this project below

- Power Management Features
- USART and SPI port
- Universal Serial Bus (USB) Features

Firstly, PIC18F46J50 has power management features such as it can choose different operating frequency from internal RC oscillator or external high frequency crystal. Power consumption can be optimized when doing different work load. Secondly, SPI port for fast speed data transmission between microcontroller and NAND Flash. In addition, UART port for wireless communication through Bluetooth module. Last but not least, compare to other microcontrollers, incorporating a fully-

featured USB communications module with a built-in transceiver that is compliant with the USB Specification Revision 2.0. The module supports both low-speed and full-speed communication for all supported data transfer types. This function can help LPCR board to communicate with computer easier and faster.

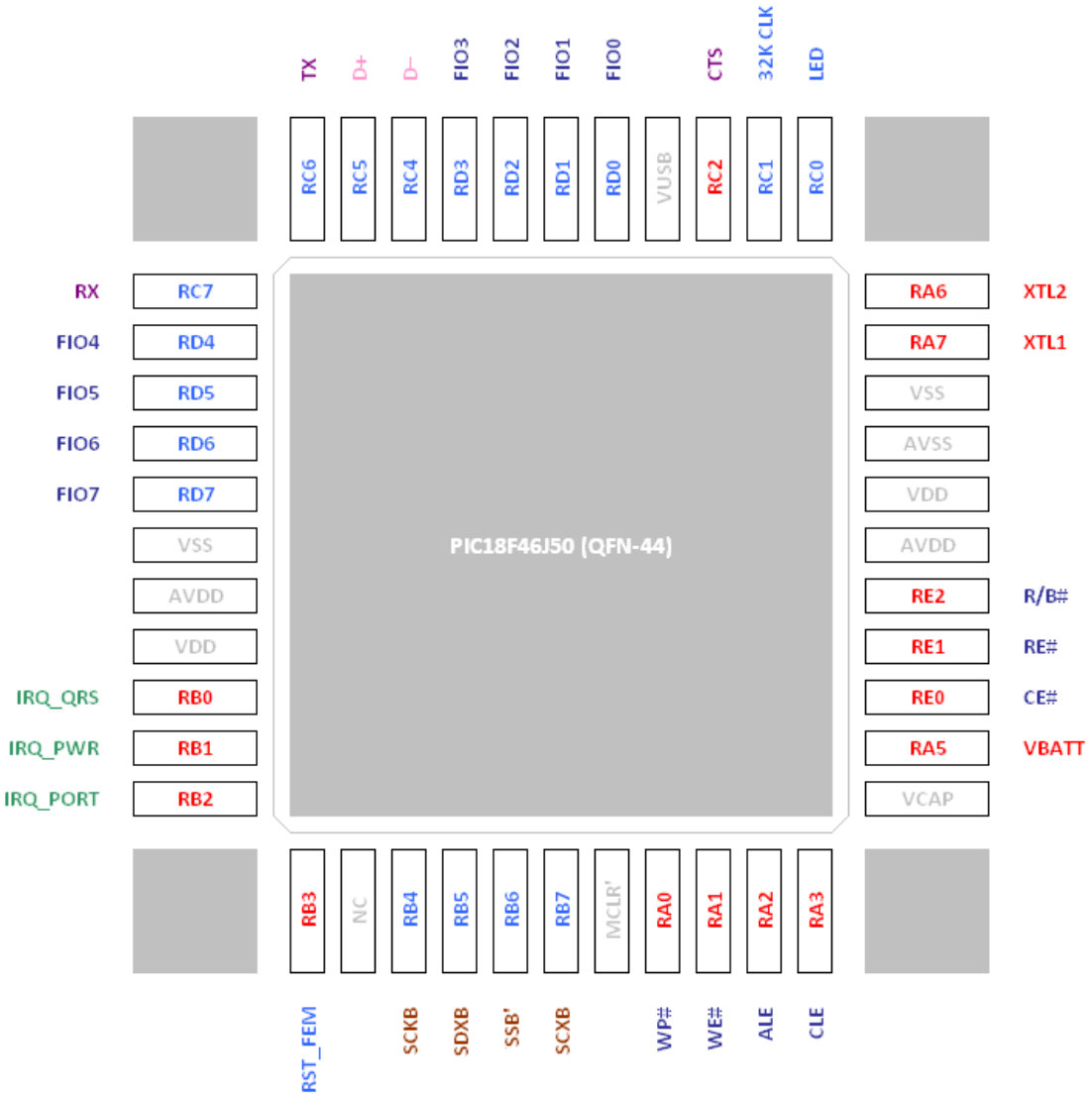


Figure 4.5: Pin configuration of PIC18F46J50 in LPCR system

In Figure 4.5, the details of the pin configuration show how PIC18F46J50 controls other components in LPCR system. These 44 pins outside PIC18F46J50 microcontroller are distinguished in different colors for their connection to different

components. The red color pins are analog or analog capable components. Light blue bins are for LED and other digital components. Green pins for interrupt pins from BMDAV7 ECG acquisition chip. Purple pins are connected to UART Port for Bluetooth module. Dark blue bins are connected to NAND Flash Memory. Brown pins are for QRS SPI interface. Lastly, the pink one is for USB port connection.

4.2.2 BMDAV7 ECG Acquisition Chip

In this project, BMDAV7 ECG acquisition chip [14] is used to collect ECG data instead of BMDAV8. BMDAV8 chip in LPCR system is only for driving support. The BMDAV7 shown in Figure 4.6 is an ultra-low-power ECG acquisition device targeting pervasive healthcare and portable medical apparatus market. The device draws only 1.75 μA from a 1.5-V supply, and offers an energy-efficient MCU interface that facilitates low-power implementation of ECG systems. The BMDAV7 integrates a fully featured low noise acquisition module that provides band-pass input filtering, programmable amplification, and 12-bit A/D conversion; and a QRS detection module that calculates the heartbeat rate. An 8-Kb onchip SRAM buffers the captured ECG data and the corresponding heartbeat rate, which are periodically flushed to the external MCU via a standard SPI port.

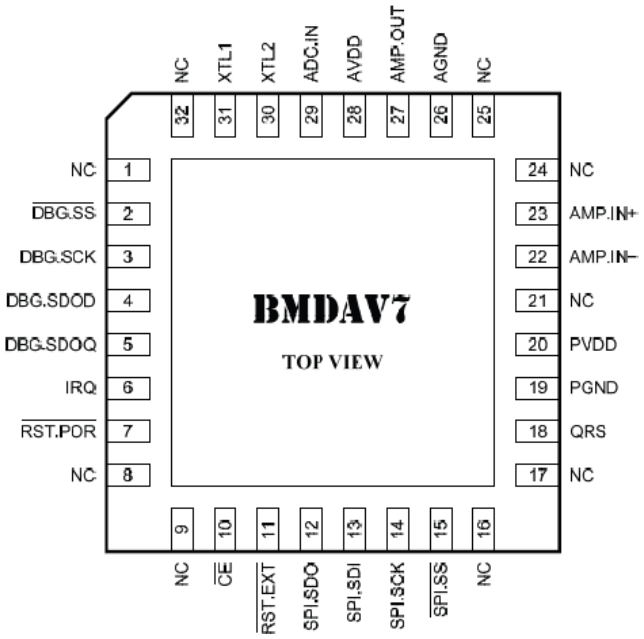


Figure 4.6: BMDAV7 ECG Acquisition Chip

Table 4.2: Comparison between BMDAV7 and BMDAV8

Property	BMDAV7	BMDAV8
Supply Voltage	1.5V	1.8 ~ 3.6 V
Technology	0.35 µm	0.35 µm
Sampling Freq	256 S/s	up to 25 kS/s
Interface	SPI slave	SPI slave
Current	1.75 µA @ 1.5V	18 µA @ 3V

From Table 4.2, the main difference between BMDAV7 and BMDAV8 is that BMDAV7 has lower sample frequency which is 256 Hz. But its power consumption is only 2.3 µW lower than BMDAV8. For LPCR system, long time measurement requires low power consumption. Hence, BMDAV7 chip is selected to collect ECG data in this project.

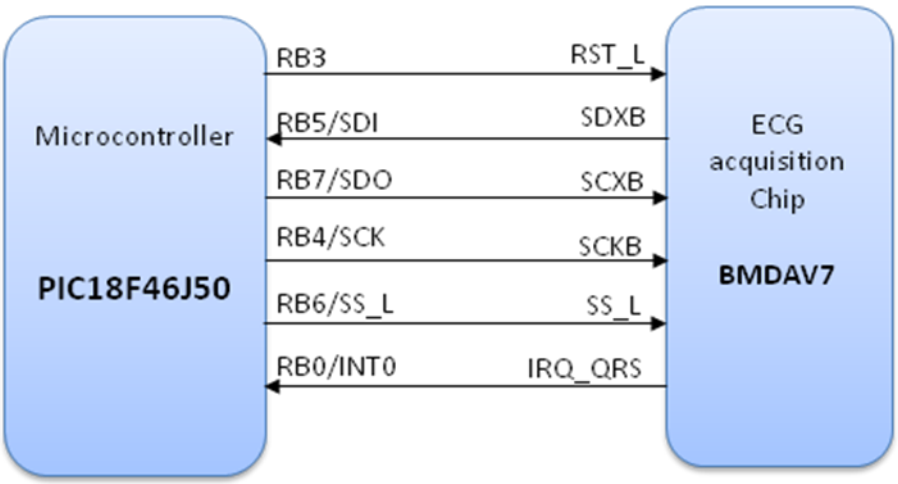


Figure 4.7: Pin configuration between BMDAV7 and PIC

Figure 4.7 shows how BMDAV7 work with microcontroller. BMDAV7 has 1024-byte internal FIFO, which could be used to store one second sampling data. The sampling frequency for BMDAV7 is 512Hz, and each sampling data is 10-bit depth. As same as BMDAV8 chip, it uses SPI communication port for controlling and transferring data. A Microcontroller (PIC18F46J50) could control BMDAV7 and receive data through the SPI port pin SCXB, SCKB, SDXB. An IRQ_QRS pin is used to inform PIC MCU that ECG data is ready. Lastly, RST_L pin can reset BMDAV7 chip function.

4.2.3 NAND Flash

There are several types of memory devices in the market such as DRAM, NAND Flash, NOR Flash and so on. In order to select appropriate devices for LPCR system, a survey is done to compare the advantages of the different memory device. The power consumption of Flash memory is 10 times less than DRAM. However, the speed of Flash is much slower than DRAM. NOR Flash can do random access, but

typically its capacity is lower than 128MB. As LPCR system needs to record twenty plus days ECG data, Flash memory with bigger capacity is first choice.

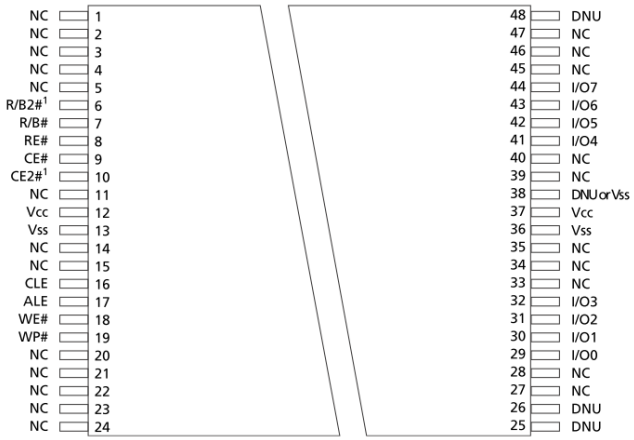


Figure 4.8 MT29F16G08DAAWP Flash chip top view

Lower power MT29F series Flash is select for this project. The reason to choose it in this project is list below

- Large Capacity : 8G/16G bit
- Low power consumption
- READ performance

  – Random READ: 25μs

  – Page READ (a special feature to perform read data for entire page in very high speed): 20ns

- WRITE performance

  – PROGRAM PAGE (Page Write, faster speed writing for one page data): 250μs

  – BLOCK ERASE: 1.5ms.

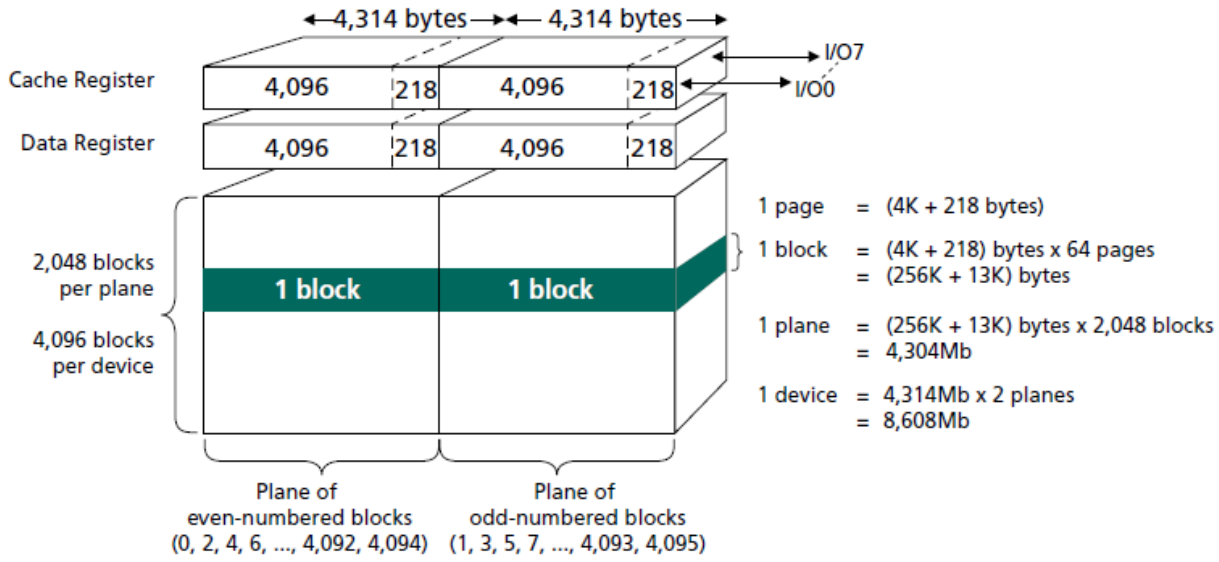


Figure 4.9: MT29F16G08DAAWP [15] Flash chip array organization

The NAND Flash used in LPCR is Micron MT29F16G08DAAWP, which has a capacity of 16Gbits. This NAND Flash has 4096 Blocks; each block contains 64 pages with every page contains 4096 bytes + 218bytes spare area. Because the sampling frequency for BMDAV7 is 512Hz and each sampling data is 10-bit depth, a 16 Gbit Flash chip can store 48 days patient ECG data.

4.2.4 Blue Giga WT12

The wireless data transmissions part is not the major objective for LPCRV1 version. This is because wireless power consumption is quite high. However, in order for further development and testing, a WT12 Blue giga module is selected for this project. Its feature is shown below.

- Bluetooth V2.1

- UART Interface for communication with device
- Low voltage supply: 3.3V
- UART: 115200,8n1

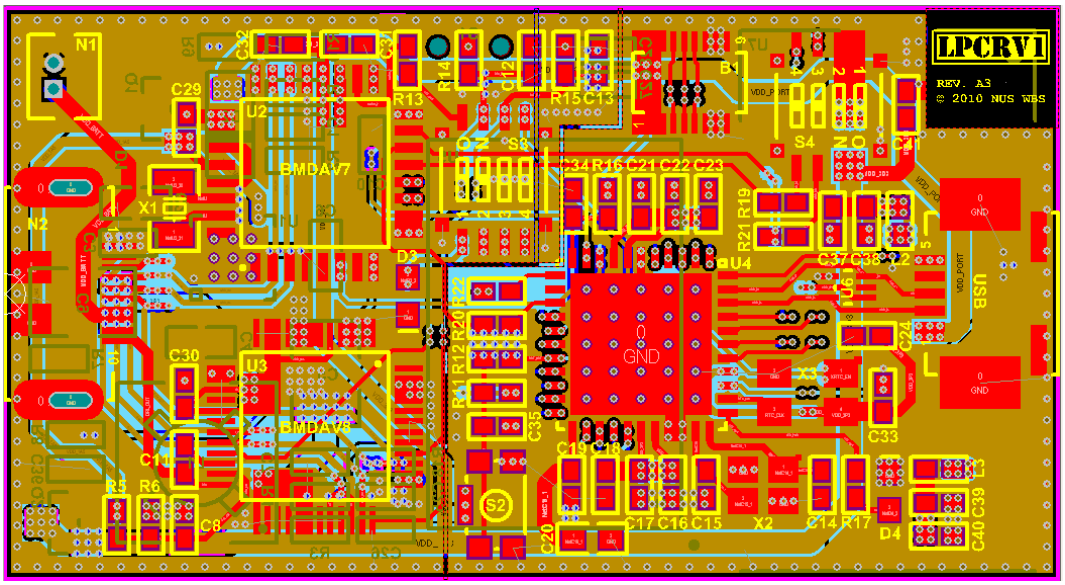


Figure 4.10: LPCRV1 PCB

As shown in Figure 4.10, finally, by integrating all the hardware components above, a 5.0cm x 2.7cm prototype PCB is implemented for LPCRV1 system.

4.3 Firmware design

The major function of LPCR system is divided into 3 parts: Data Recording part, USB data transfer part and wireless data transfer part. To make the device consume less power, different operating modes are used. When switch between different operating modes, the Microcontroller could be configured with different operating

frequencies. In LPCR system, we use two frequencies: 31 KHz and 48MHz. Heavy jobs need to be done at high frequency mode in order to complete it faster. Idle state will remain at low operating frequency to save more power.

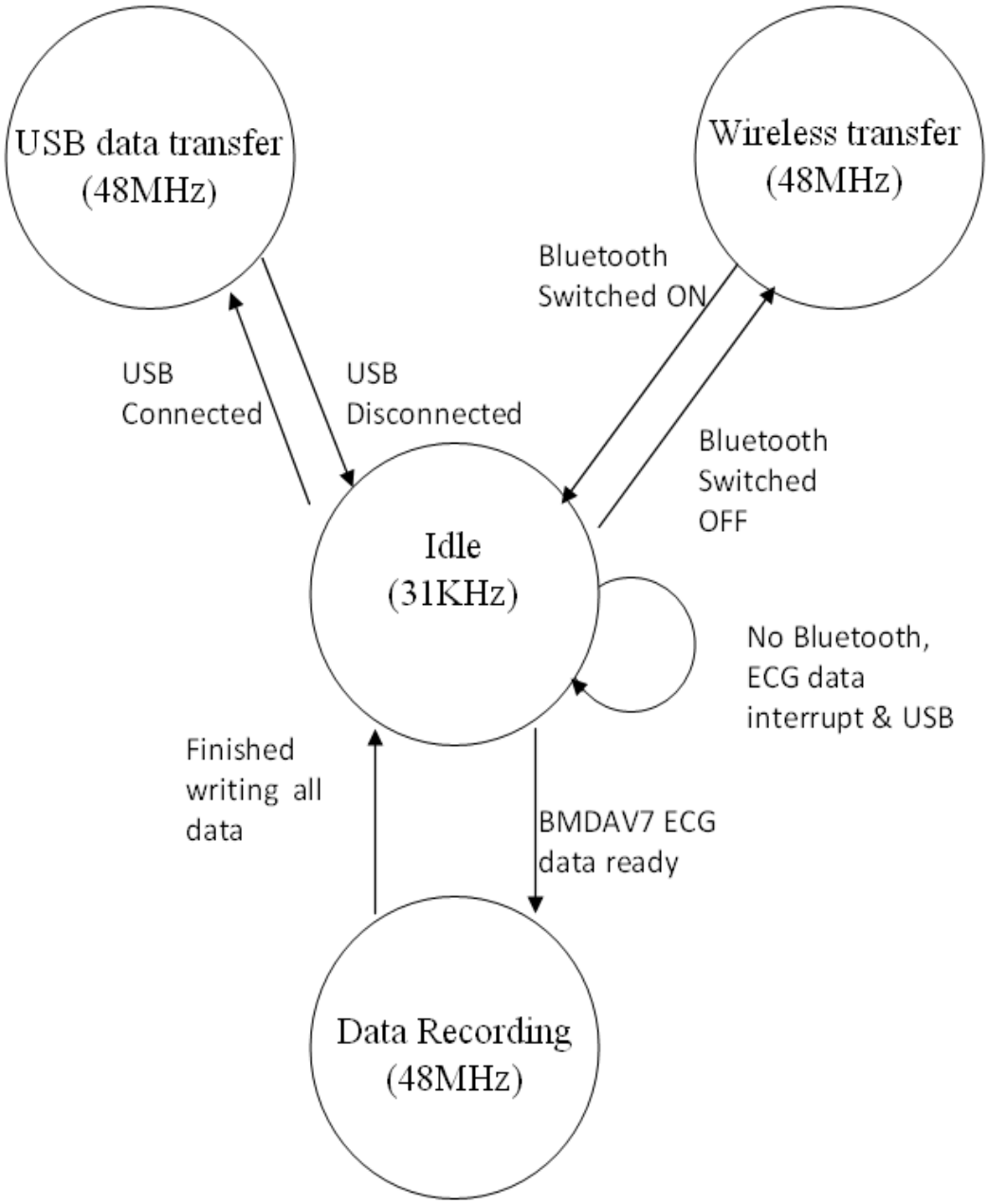


Figure 4.11: Firmware state diagram

The description for Figure above is shown below. In this work, the firmware part

related to data recording state will be explained in detail.

State: Idle

-- System Clock: 31 KHz

-- Only BMDAV7 collect ECG date

-- Low power, ~80uA

State: Data Recording

-- System Clock: 48MHz

-- Read ECG controller BMDAV7's data and store to NAND Flash

State: USB data transfer

-- System Clock: 48MHz

-- Transfer Flash data to PC

State: Wireless transfer

-- System Clock: 48MHz

-- Use Bluetooth module to transfer ECG data to PC

4.3.1 Microcontroller and BMDAV7

In data recording mode, the first objective is to let MCU and BMDAV7

communicate with each other for ECG data collection. Previously, ECG sampling rate was 256 words per second, BMDAV7 has1024-byte internal FIFO, which could be used to store 2 second sampling data. Hence, V7 will communicate with MCU every 2 seconds. Their communication is set by these control signals. BMDAV7 uses SPI communication port for controlling and transferring data

As shown in the picture below, the data receiving from BMDAV7 by SPI port is in two bytes format. The first 10 bits indicate the ECG data. The Q7 down to Q4 are four MSB data of QRS, Q3 down to Q1 will be received in the next two cycles. F1 and F0 are dummy data.

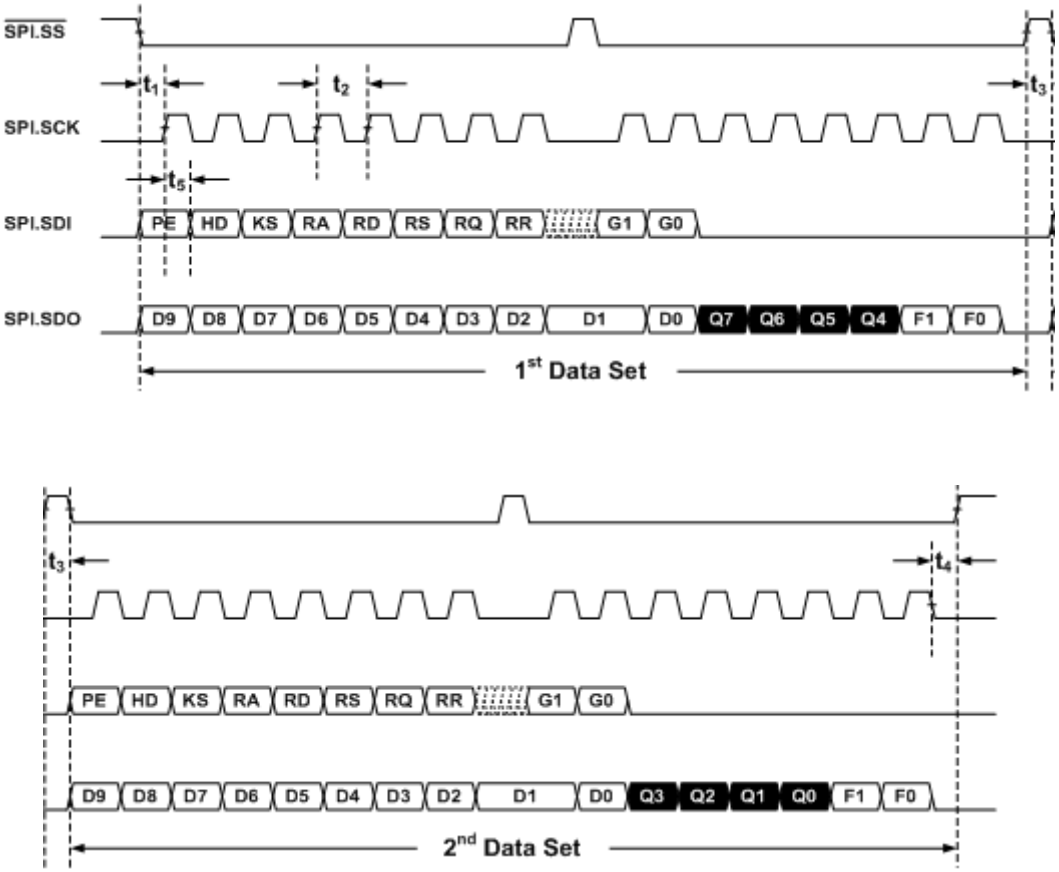


Figure 4.12: ECG control signals

In order to start the communication successfully, there are two steps: initialization and receiving data. These are done by the control signal set in the firmware; Table 4.1 and Table 4.2 show the detailed setting.

Table 4.3: BMDAV7 control bits

Bit	Description
PE	Program Enable
HD	SRAM Hold (Hold SRAM read pointer)
KS	Clock Selection (SPI's clock or crystal clock)
	1: Uses crystal clock
	0: Uses SPI clock (for transmitting data)
R1	Amplifier Reset
R2	ADC Reset (Active low)
R3	SPI Reset (Active low)
R4	QRS Reset (Active low)
R5	SRAM Reset (Active low)
G1,G0	Amplifier Gain control bits (45, 49, 53, 60dB)

Table 4.4: BMDAV7 status bits

Bit	Description
D9 ~ D0	ECG data value
Q7 ~ Q0	Average Heart Rate value
F1 ~ F0	00 FIFO status is EMPTY 01 FIFO status is READY 10 FIFO status is CRITICAL 11 FIFO status is OVERFLOW

To initialize V7, Microcontroller sends 0xE0 and 0x00 via SPI port first, the data means

- Program Enabled
- SRAM pointer hold

- Clock: Using Crystal clock for transmitting data
- Resets Amplifier, ADC, SPI, QRS and SRAM
- Amplifier's gain set to default (00)

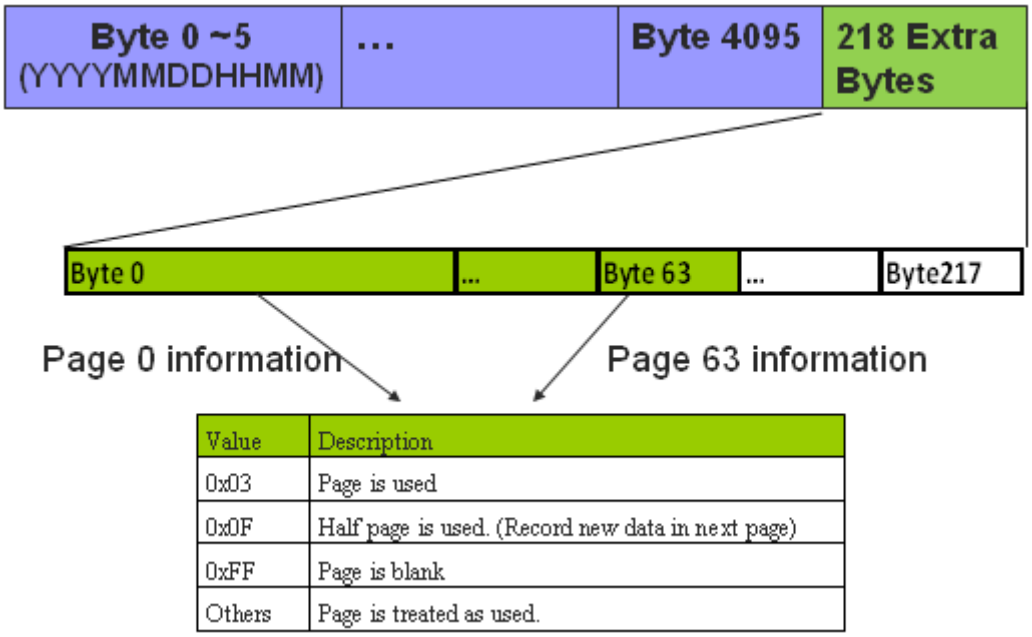
Then the next step is receiving ECG data, Microcontroller sends 0x9F and 0x00 this time means: SRAM is un-hold, Use SPI clock to transmitting data and Amplifier, ADC, SPI, QRS and SRAM do not reset. Based on the setting above, ECG signal can be transferred to MCU though SPI interface successfully.

4.3.2 Microcontroller and FLASH

The second job in data recording mode is to save ECG record in a memory device. The firmware code of this part is shown in Appendix 2. In order to write and read data into Flash memory, file structure is needed to be design first. File structure is how data arranged in memory. An effective structure can help entire system to save the time spent on reading and writing data. As introduced in section 4.2.3, FLASH MT29F has special feature page read and page write function. It can help users to read the entire page in very high speed. Hence a good file structure with clear page status information can help system reduce a lot of time. Which also means power consumption can be reduced.

The Simple File System (SFS) Structure for LPCR system is explained below. As shown in Figure 4.9, Flash MT29F16G08DAAWP has 2048 blocks. Each block contains 64 pages. Each page has 4096 bytes + 218 bytes (Extra Bytes Region). The Figure 4.13 shows the detail of page information. In LPCR system, the first byte 0 to

5 of each page is used to store the timing information of patient's recording. The next 4090 bytes are dedicated for ECG data. Then, Extra bytes area is used for save a clear page information. With this page information, the feature page read and page write of MT29F16G08DAAWP flash chip can be activated.



Value	Description
0x03	Page is used
0x0F	Half page is used. (Record new data in next page)
0xFF	Page is blank
Others	Page is treated as used.

Figure 4.13: File structure of FLASH memory

For each block, only Extra Date Region of page 0 will be used to store special page information. For example: if data read from byte 1 of extra 218 byes is 0x0F, it indicates this block has already been used more than half. The data written into the Flash will start next block. LPCR system will keep on checking this process in order to prevent overwriting problem. By implementing this quick access to page information method, PIC microcontroller can read and write ECG data to Flash memory in very high speed.

The communication between microcontroller PIC18F46J50 and Flash MT29F16G08DAAWP is controlled by the pins in the Figure below. The Table below shows the detail function's of each control pins. CLE, ALE, CE#, RE# and WE# are the most often control signal used in the project. I/O [0~7] are the data bus between 2 chips, and they can transmit the data simultaneously.

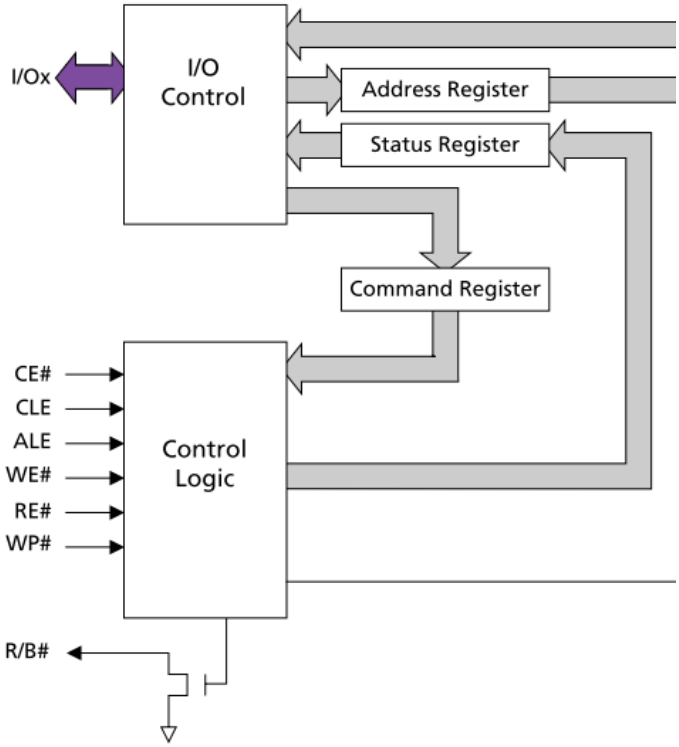


Figure 4.14: MCU control block

Table 4.5: Control bits for FLASH reading and writing

Pin	Direction	Description
ALE	Input	Address latch enable
CE#	Input	Chip enable, active low
CLE	Input	Command latch enable
RE#	Input	Read enable, active low
WE#	Input	Write enable, active low
WP#	Input	Write protect, active low
I/O[0~7]	I/O	Data inputs/outputs
R/B#	Output	Ready / Busy

After initializing the Flash chip with control pins above, Read and Write to Flash is able to start. To read one page data from the NAND Flash array, first, write the 00h command to the command register, then write 5 ADDRESS cycles, and conclude with the 30h command. To determine the progress of the data transfer from the NAND Flash array to the data register (tR), monitor the R/B# signal; or alternately, issue a READ STATUS (70h) command. If the READ STATUS command is used to monitor the data transfer, the user must re-issue the READ (00h) command to receive data output from the data register. After the READ command has been re-issued, pulsing the RE# line will result in outputting data, starting from the initial column address.

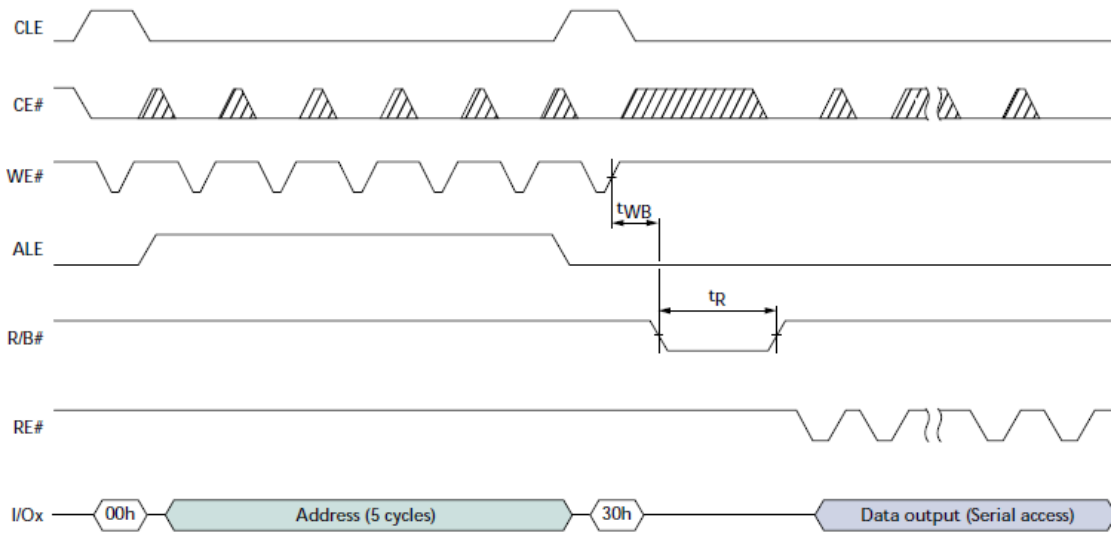


Figure 4.15: Flash read setting

A serial page read sequence outputs a complete page of data. After 30h is written, the page data is transferred to the data register, and R/B# goes LOW during the transfer. When the transfer to the data register is complete, R/B# returns HIGH. At this point, data can be read from the device. The process to write data to Flash is more or less the same.

During ECG data recording, clock switching method is applied. Switch System Clock is a method to save the Flash data reading and writing power. When the system clock is set to 31 KHz, LPCR is in Idle mode (low power). On the other hand, when microcontroller is going to make Flash Data recording, system clock will adjust to 48MHz. The faster the data transmission, the less the power will be cost for reading and writing. With well optimized algorithm, the firmware could operate the system with very low current: experiment measurement shows the current consumption is less than 1.7mA in average.

4.4 Graphical user Interface

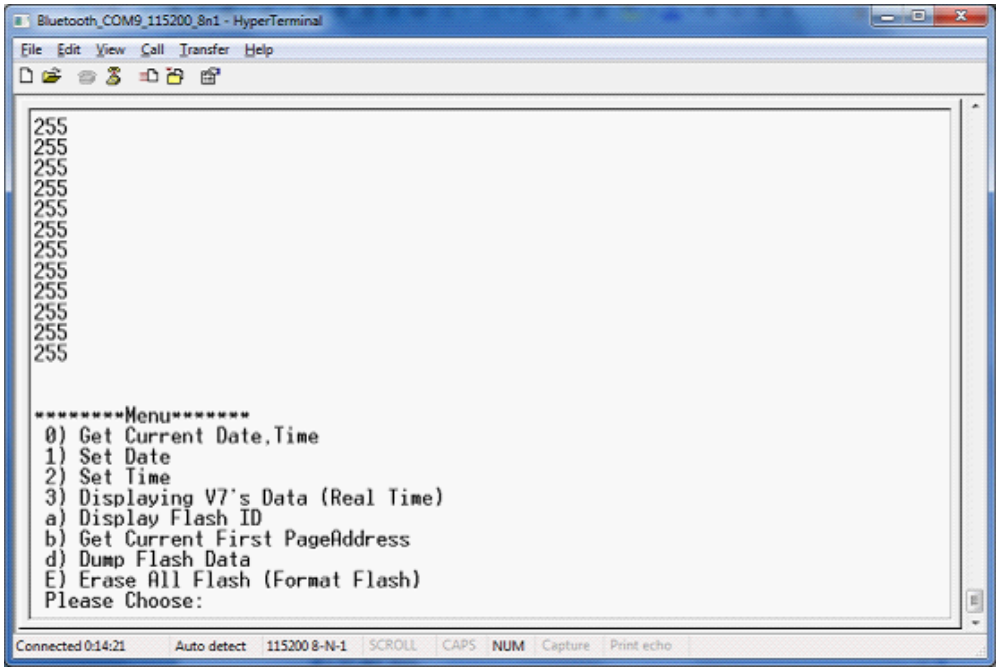


Figure 4.16: Graphical user interface

The graphical user interface for LPCRV1 system is using hyper terminal on PC. Its display is simple but the function is completed. GUI of LPCR is for user to collect ECG data stored in Flash. User can choose the data format to be decimal or hex. In addition, users can use the function shown in the Figure 4.16 to some basic setting suck like to Get current date and Time information, set Time information, Display BMDAV7's ECG data, Erase all data stored in Flash and so on.

4.5 Design verification

LPCRV1 is the first version of this project. The major objective of this version is to prove that the proposed long time playing idea can work. Hence, there is no professional clinical trial done for LPCR system in this version. On the other hand, an ECG simulator testing, volunteering testing and long time battery testing is done here to prove the system accuracy and reliability.

4.5.1 ECG simulator testing

Figure 4.17: ECG simulator

Firstly, a commercial ECG simulator shown in Figure 4.17 is selected for this testing. During the three hour testing, ECG simulator will generate 3 different QRS signals or average heart rate ECG signal in each hour. LPCR system is used to capture the signal and store it to Flash memory. The testing result is shown in Figure 4.18; the 30Hz, 60Hz and 90Hz ECG signal is very clear from LPCR system.

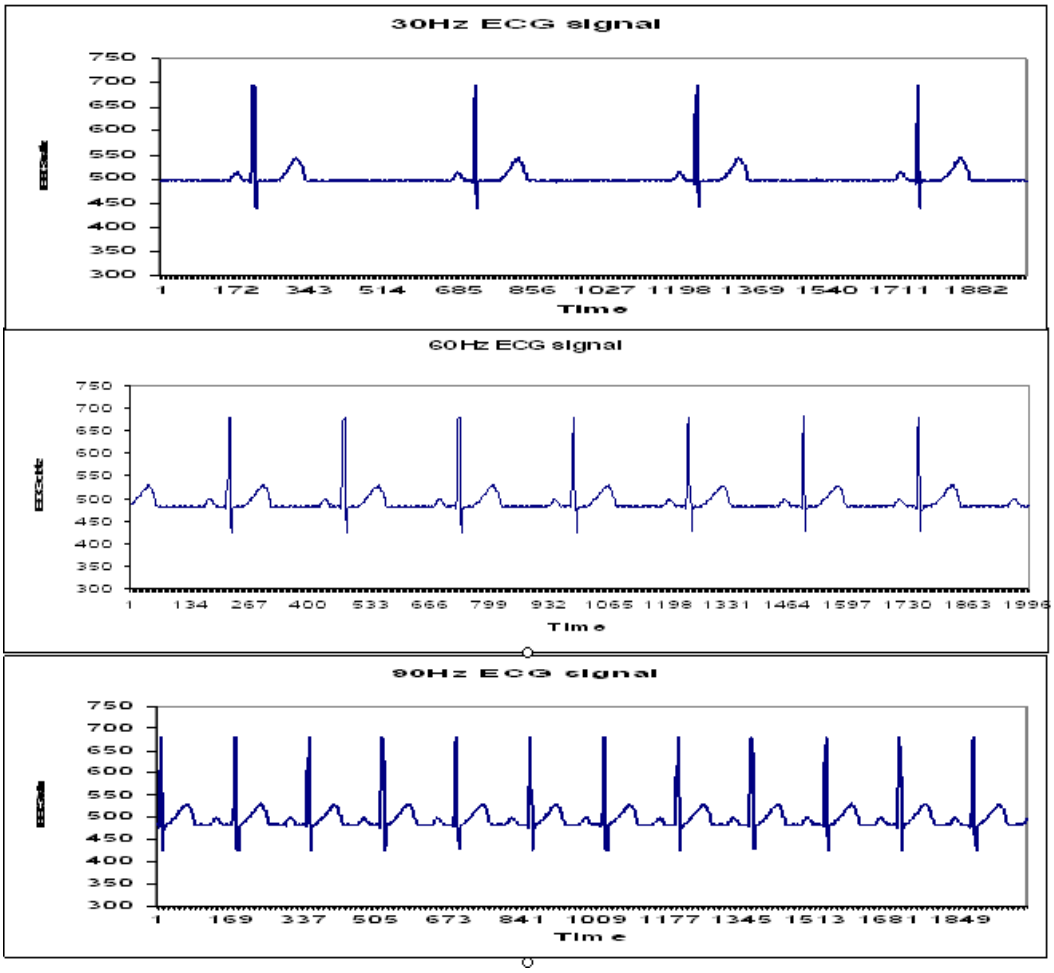


Figure 4.18: 30bpm, 60bpm, 90bpm ECG signal Simulation result

Table 4.4 shows a comparison of ECG simulator hear rate setting and LPCR testing result. They are almost the same during this 3 hours simulation test.

Table 4.6: Testing result For Average heart rate

ECG Simulator Heart Rate setting	LPCR testing Result
30 bpm	30 bpm
60 bpm	60 bpm
90 bpm	89-90 bpm

4.5.2 Volunteer testing

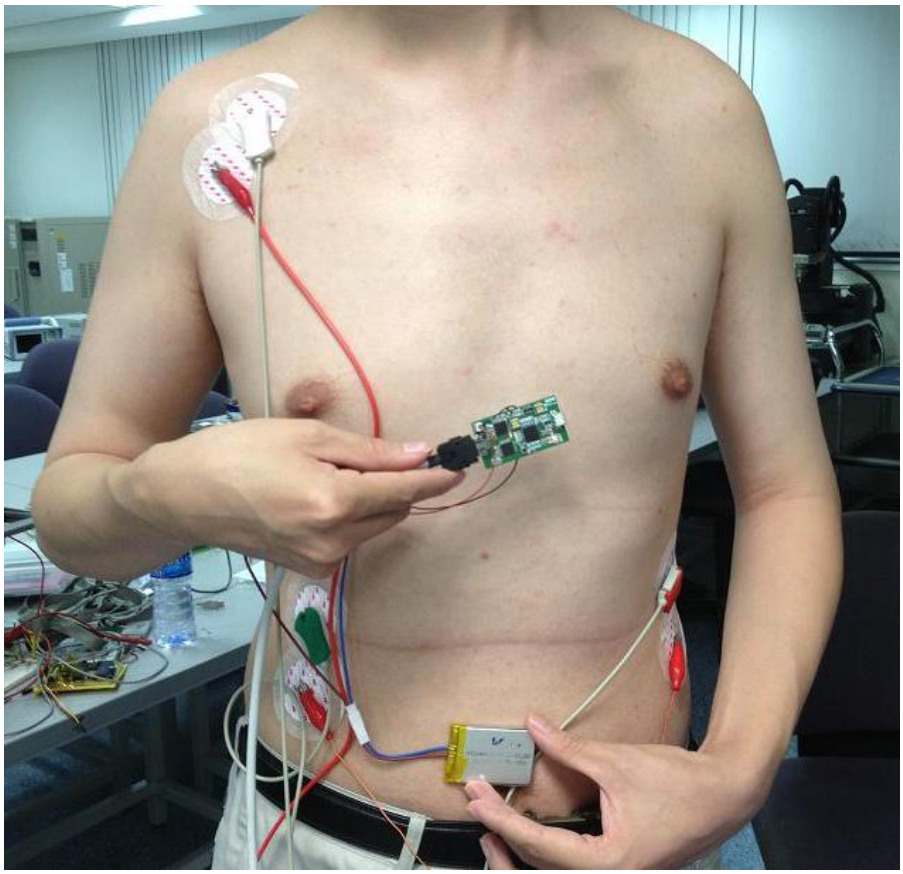


Figure 4.19: The positions of the LPCRV1 and Welch allyn device

Moreover, LPCR device has been verified in lab tests on volunteer. The prototype

device and a commercial reference device were simultaneously used to collect 5 minutes ECG from the subjects. We used WelchAllyn cardioperfect ECG measurement system as reference. To establish the short term equivalence of ECG obtained from both the devices, the signals from different subjects were plotted and compared side-by-side as shown in Figure 4.19. Since the ECG signal was collected from the same lead, both the signals should appear roughly similar. The result is shown in the Figure below. The signal on the top is from Welch Allyn device and signal at bottom is from LPCR system.

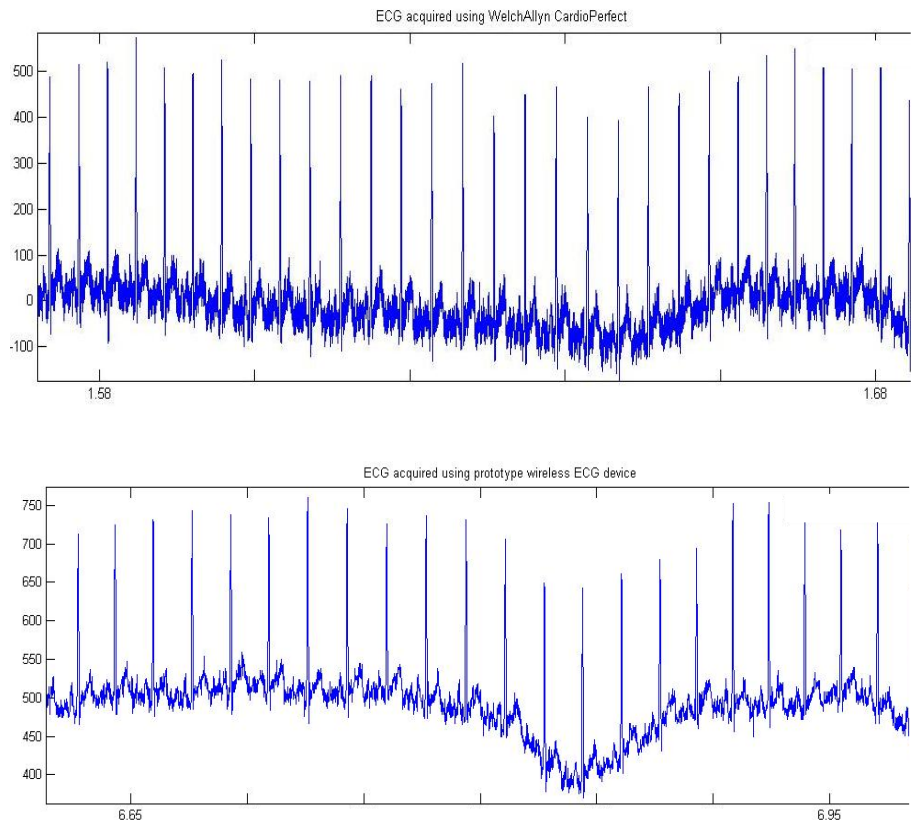


Figure 4.20 Volunteer test result from Welch Allyn device and LPCR system

It's obvious that the two ECG signal trends from two devices look the same. The average heart rate from reference device Welch Allyn cardio perfect device is 93 bpm.

LPCR system shows the Avg. Heart rate is 92 bpm. They are almost the same too. In addition, we also computed the RR interval for every beat in the ECG signal. The average difference in RR interval obtained using both the devices is found to be less than 2% of the reference device. The histograms showing the RR interval for both data sets are shown in Figure below. The LPCR system accuracy has been proven.

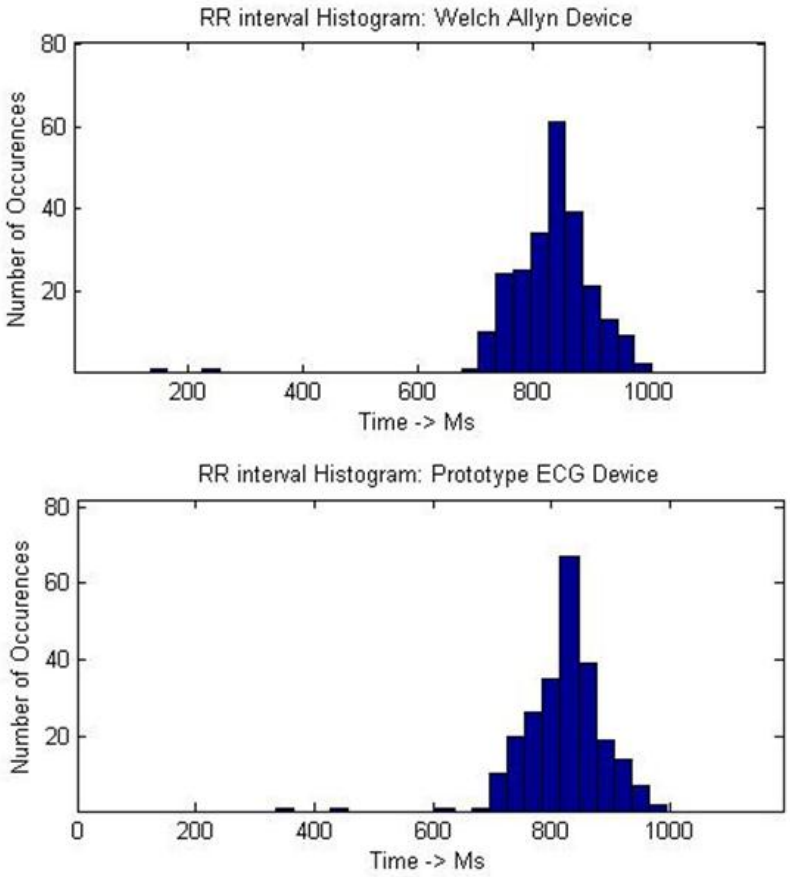


Figure 4.21 RR Interval histograms: LPCR system Vs Reference Device

4.5.3 Long time battery testing

To prove the LPCR system's long time recording reliability, we conduct a 30 days

continuous running long time battery testing. The setting is list below

- Testing with single 3.7V 650 mAH Hi-charge Battery without recharge for 30 days
- ECG simulator is used to generate ECG signal. It continuously changes 30, 60, 90 Hz ECG signal every 12 hours to perform as a human being.
- Sample data are collect every 12 hours to prove LPCR system working correctly
- Average current consumption is 1.7 mA during testing.

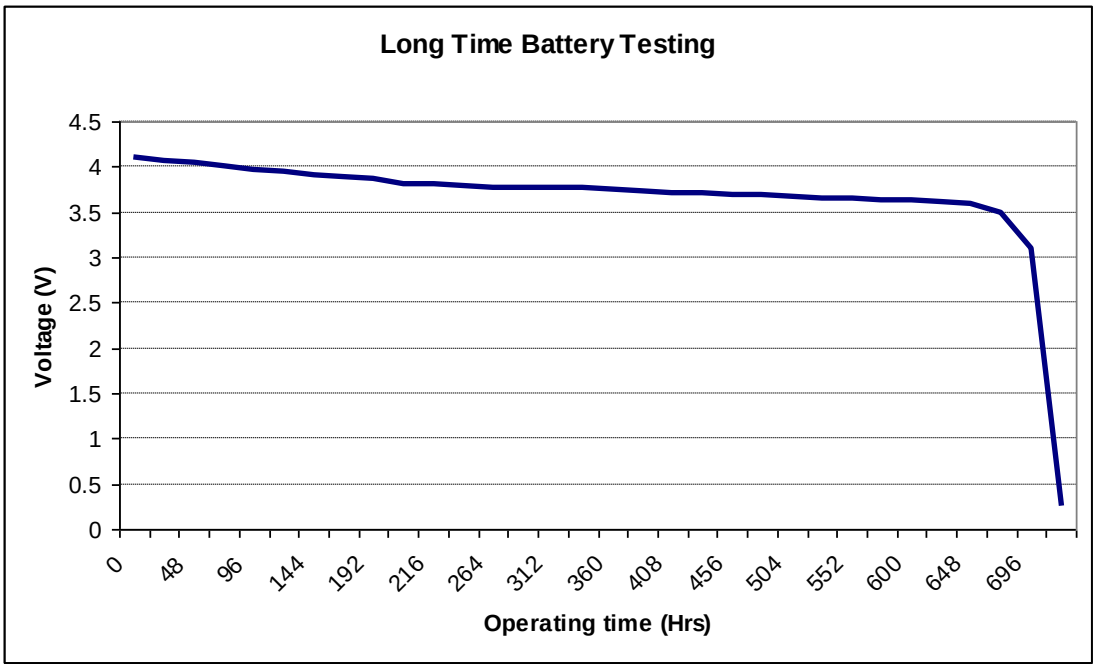


Figure 4.22 Long time battery testing result (Battery voltage VS time)

Figure 4.22 is the long time battery testing result. It's clear that the battery power

can last for 718 hours which is almost 30 days. At the first 696 hours, the voltage level of battery drops slowly. A sharp discharge is shown when the voltage level drops to 3.4V. This phenomenon matches Li-ion battery discharge characteristic. As a result, LPCR system can continue monitor ECG signal for more than 30 days, so long time reliability has been proven.

In conclusion, Long Playing Cardio Recording system is designed for 48 days long term ECG data recording, and it is also a wearable device. It receives data from ultra-low power ECG acquisition chip. The data is stored into to a 16G bit NAND flash. The system current consumption could be less than 1.7mA from a 3.7V 650mAH Li-ion battery so it can last for 30days.

Chapter 5

Wearable ECG system performance comparison

In previous chapters, two energy efficient wearable ECG monitoring devices are presented. In order to see the performance of Wireless ECG Plaster and Long Playing Cardio Recording system, a comparison between this work and two other existing designs are summarized in the table 5.1

Table 5.1: Comparison between other ECG monitoring systems

Property	LPCR system	Wireless ECG Plaster	I-Jan Wang's work[7]	Jose's work[6]
PCB Size	5 cm x 2.7cm	2.8cm x 2.4cm	4 cm x 2.5 cm	30cm x 10cm x 3cm (device dimension)
Electrode	Tradition 3 leads	Plaster	Dry Polymer based	Tradition ECG leads
Memory	16 Gbit (48 days)	NA	NA	8 Gbit (7 days)
Sample rate	256 S/s	up to 25K S/s	500 S/s	500 S/s
Voltage supply	3.7V 650mAH	3.3V 650mAH	3.3V 1100mAH	9V 300mAh
Current Consumption	1.7 mA for data recording	25 mA	31 mA	6 mA
Continuous running time	30 days	26 hours	24 hours	24 hours
Accuracy	> 98 %	> 99%	> 99%	>98 %

Jose's work is an Ambulatory Electrocardiogram Recorder. It use 9V power supply and can continuously record 7 days result. But it's more like tradition holter which is heavy and big. I-Jan Wang's work proposed a wearable mobile electrocardiogram monitoring system. The feature of this system is to use dry foam electrodes. However, its power consumption is the highest among all four designs. Compared to other ECG monitoring systems, Wireless ECG Plaster is the smallest wearable ECG system (2.8cm x 2.4cm). It uses an ECG electrode embedded with a Plaster substrate in the design. Patient can feel more comfortable when they are doing ECG monitoring treatment. Wireless ECG Plaster doesn't have memory device to store ECG data, so it has to send data to a gate though Zigbee RF transceiver. Its' sample rate is the also the highest. Two clinical trials have proved that its system accuracy is quite good. On the other hand, LPCR system's power performance is the best. It can continuously run for 30 days and capable to store 48 days ECG data at one time. Though its sample rate is lower than others, its system accuracy is not so bad. In conclusion, the objective of design Energy efficient wearable ECG system has been reached.

Chapter 6

Asynchronous 8051 design

In order to further reduce power consumption of wearable ECG system, certain low power techniques are needed to implement in the hardware component. In this chapter, a new version asynchronous 8051 microcontroller is introduced to improve the power performance of wearable ECG system in the future.

6.1 Introduction

From previous wearable ECG systems chapters, it's not difficult to find low power consumption is one of the crucial concerns for portable ECG recording system design. Otherwise, the battery cannot last very long time. The microcontroller, which is a significant source of power consumption for central control block, therefore should have the desirable characteristic of low-power consumption. Also as the microcontroller may need to operate in different modes through adjusting the supply voltage level, it should be functional in a wide range of supply voltage. Hence, an

asynchronous microcontroller adopting the dual-rail four phase protocols is chosen to be the desired microcontroller for ECG recording system. In the year 2010, NUS M.ENG student Xue Chao has done a good research on dual-rail four phase asynchronous 8051 microcontroller [16]. The low power async 8051 microcontroller proposed in this chapter is an improved version based on his work in order to save Wearable ECG system's power.

6.1.1 Synchronous 8051 microcontroller

The asynchronous microcontroller presented in this chapter follows the structure of a standard synchronous 8051 microcontroller shown in Figure 6.1.

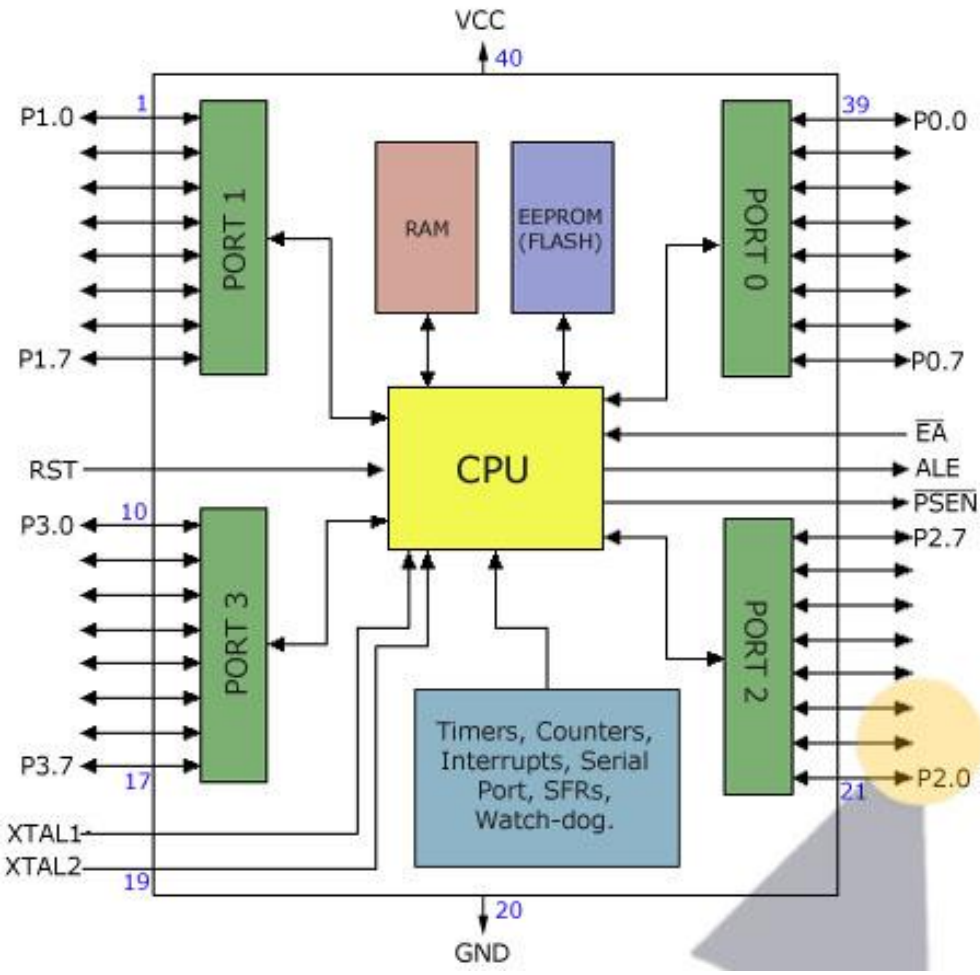


Figure 6.1: 8051 Microcontroller block diagram

The Intel MCS-51 (commonly referred to as 8051) is Harvard architecture, single chip microcontroller series which was developed by Intel in 1980 for use in embedded systems. One of the major reasons to use 8051 microcontroller architecture is because 8051 was widely used in the world. Developer can design the firmware on previous wearable ECG system easily. Its main features are below.

- 8-bit data bus
- 16-bit address bus
- On-chip 128 byte of internal RAM
- On-chip 4 KB of internal ROM
- Four general purpose I/O ports each of 8 bit wide
- On-chip programmable fully duplex USART serial port
- Two on-chip 16-bit timers
- Two-level priority interrupt handling
- Over 200 instruction set

6.1.2 Asynchronous circuit design flow

The design flow starts from hardware description language (HDL) coding. The asynchronous 8051 core adopts Balsa and the 8051 peripheral function block is uses verilog. The Synopsys Design Compiler can optimize and compile these HDL files into unified Verilog gate-level netlist, which is then imported into the Cadence Encounter together with the LEF (library exchange format) file for automatic P&R

(placement and routing). A GDS (graphical database system) file of the asynchronous 8051 microcontroller is exported from the SOC Encounter into the Cadence Virtuoso framework. After performing LPE (layout parasitic extraction), the SPICE netlist is passed to Modelsim for final post-layout simulation verification.

6.2 Architecture of the Asynchronous 8051

6.2.1 Overview of Asynchronous 8051

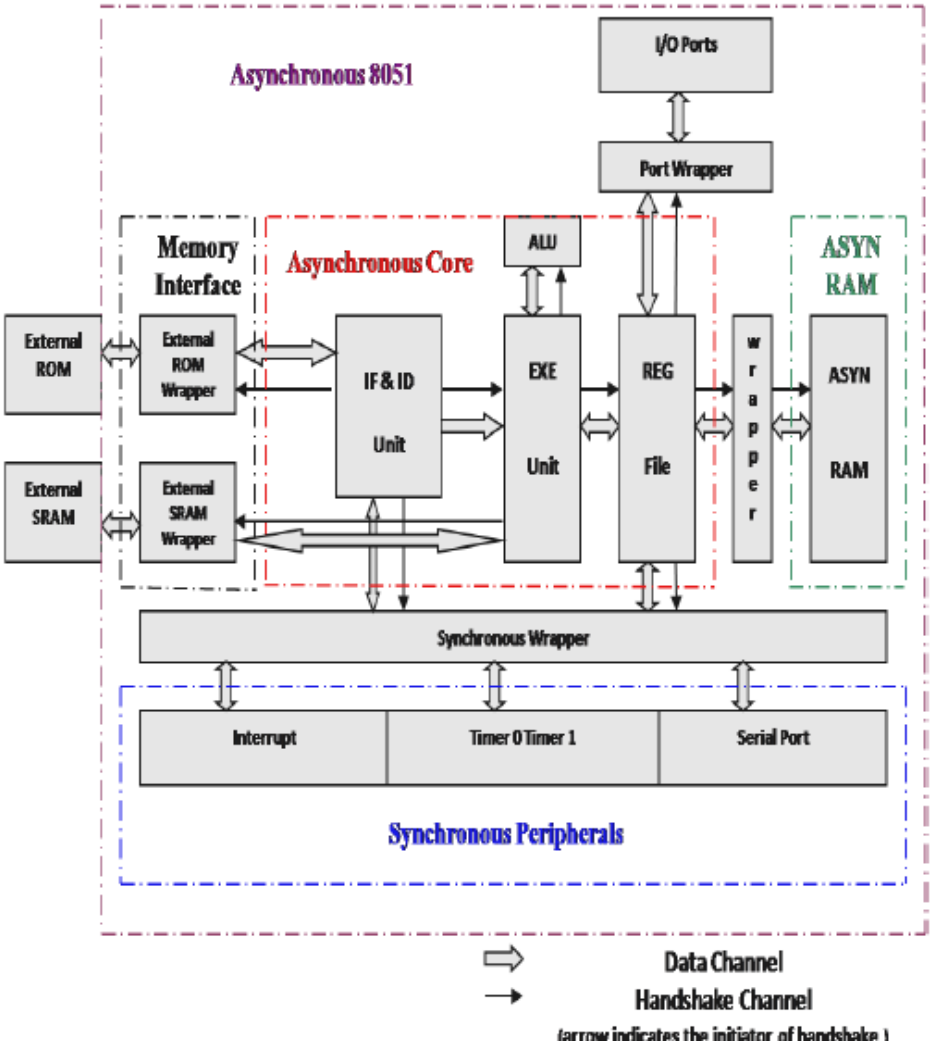


Figure 6.2: Async 8051 Microcontroller [16]

Figure 6.2 is asynchronous 8051 microcontroller's block diagram. There are five major blocks in the proposed asynchronous 8051: asynchronous core, synchronous peripherals, external memory interface, I/O ports and asynchronous internal All the signals within the asynchronous core are dual-rail in nature. For the communication between Asynchronous part and the outside part, wrapper blocks are designed around the asynchronous core to perform the data conversion between dual-rail data and single-rail data.

In order to perform as same function of as synchronous 8051 microcontroller, it has to match all kinds of instruction sets which sync 8051 have. Hence, a proper design of 8051 Asynchronous core is needed.

6.2.2 8051 Asynchronous core

The Asynchronous core designed is the central processing part of Asynchronous microcontroller. There are four major sub-blocks inside the asynchronous core: the instructions fetch and decode unit (IF & ID), the execution unit (EXE), the ALU (ALU), and the register file (REG File) unit as shown in Figure 6.3.

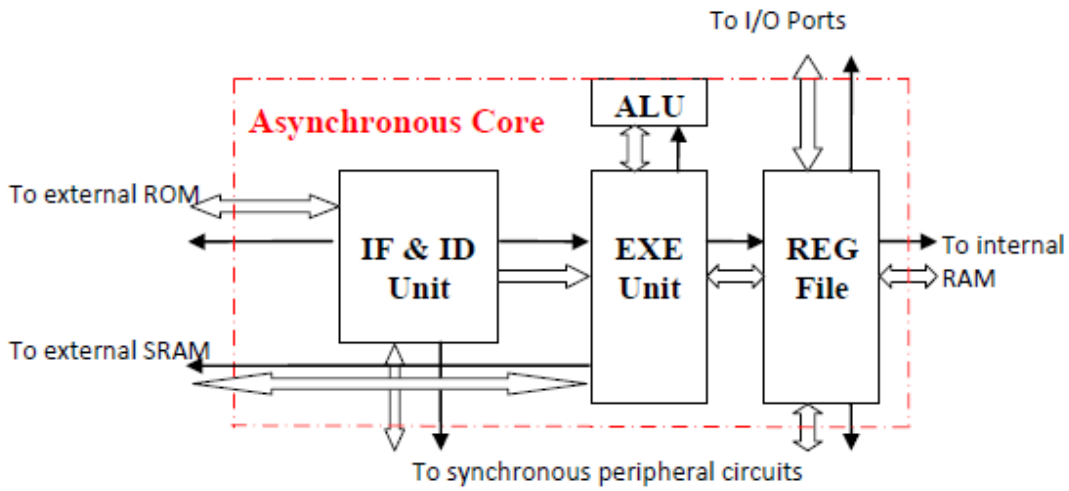


Figure 6.3: Async 8051 core

Firstly, the instructions fetch and decode unit is for fetching and decoding of the instruction bytes and the checking of the current interrupt status coming from the interrupt controller of the synchronous peripheral. Secondly, the instruction execution unit is a structure corresponds with all the instruction set. Thirdly, Register File Unit controls the access to the Special Function Registers (SFR) space and internal RAM of the 8051. Lastly, ALU unit is a dedicated block for arithmetic and logical operations. The ALU unit is a case structure with 15 valid sets, such like ADD, SUB, AND, MUL, DIV and so on. The multiplication (MUL) and division (DIV) operations blocks are specially designed since the Balsa framework does not have operators that support integer variable multiplication and division. With these two blocks, the proposed Async core can fully react as sync 8051 microcontroller.

The objective of previous work is to improve the speed of unit execution. Hence, its structure is three-stage pipelined design. In this structure, the interrupt checking block is taking out of the Instruction Fetch and Decode block to form a standalone

block which executes in parallel with the other synchronous peripherals block. It can reduce the time which asynchronous core has to wait one clock period for the valid incoming interrupt status data. However, this structure too complex and require insertion of balanced buffer trees. Though the operating speed is significantly improved, the power consumption is increased and it can't fully work when an interrupt is raised. Hence, a simple asynchronous core structure is needed.

This work focus on two changes made within the Balsa framework. The first change is to add in the Multiplication and Division operations for the ALU. The second change uses a simple structure to maintain low power.

In wearable ECG system, power consumption is the primary concern. Hence, the structure for new version async core is Non-pipelined with MUL and DIV structure. Firstly, as shown in Figure 6.3, Non pipeline structure is used here, Instruction Fetch and Decode stage, Instruction Execution stage and interrupt checking block pipeline structure is removed. The design is the simplest design because it doesn't have parallel job to cope with. Though the operating speed is slow, the power performance is better. Secondly, MUL is dedicated to handle the integer multiplication operation. It is composed of a series of add and shift operations to derive the resultant product value. The DIV for integer division operation is formed by sub and shift operation. After integrating DIV and MUL function block into no pipeline structure core, the new version of low power asynchronous 8051 microcontroller core is completed. The detail balsa code for this asynchronous core is shown in Appendix 3.

6.3 Simulation Result

In order to prove the new asynchronous 8051 microcontroller, a Nanosim post-layout simulation is conducted in this part. The whole simulation setting is from previous design. There are two performance indicationg parameter need to be introduced in this simulation testing. One parameter is called Million Instruction per Second (MIPS) which represents the number of instruction that can be completed within one second. The other parameter presents the energy consumption of the asynchronous core to complete one instruction and it is denoted by "Energy (pJ)/Instrn" in the tables. These two parameters are widely used in microcontroller analysis.

After testing, the new version Asynchronous 8051 microcontroller can run at 2.1 MIPS at 3.3V supply and 0.22 MIPS at 1.0V supply. The energy consumed per instruction is about 165pJ and power consumption is about 40 μW at 1.0V supply. MUL and DIV block can work properly under different voltage supply.

Table 6.1: Comparison with other existing designs at 1.1V 0.35μm

Design	MIPS	μW	pJ/Instrn
This work	0.34	67	194
Previous design	0.62	121	203
Sync80c51 [17]	1.3	1.480	1100
A8051 [18]	0.6	70	130

Compare this work and previous design, the MIPS drops a lot because of all pipeline design has been removed. Hence, the speed of unit execution will be reduced.

But the good phenomenon is the power consumption of this work is 55.37% of previous design. As the most important concern for wearable ECG system is the low power consumption, the direction of this research is correct. Sync80c51 design has the best MIPS performance, but its power consumption is also the largest among these 4 designs. The “A8051” has the lowest energy consumption per instruction due to its bundled data approach. But its voltage supply range is fixed. It’s not so robust compare to this work which uses four-phase dual-rail protocol.

The major objective for this new version asynchronous 8051 microcontroller is to further reduce the power consumption of wearable ECG system. Compared this work to PIC and MSP microcontroller in Chapter 3 and 4, its power consumption is the lowest. The objective has been reached.

Chapter 7

Conclusion

This work has presented design and implementation of energy efficient wearable ECG system. Firstly, a wireless ECG plaster for real-time cardiac health monitoring is presented. The proposed device is wearable, light weight and can wirelessly transfer the patient's ECG signal to a remote monitoring station, where it can be analyzed in detail. The device has a battery life of around 26 hours using a 650mAH rechargeable Lithium Ion battery while performing continuous ECG recording. The proposed device has been compared with a reference ECG Holter for accuracy. The results show that the accuracy of ECG acquisition using the proposed device is bigger than 99%, and the variation in key ECG parameters obtained from proposed device and the reference device is acceptable for clinical usage. Also the stability of the device for long-term operation has been checked from a continuous 40-hour ECG recording trial.

Secondly, a Long Playing Cardio Recording system is designed for 48 days ECG data recording, and is also a wearable device. It receives data from ECG controller (V7) and stores the data to a large capacity NAND flash with 98% accuracy. PC could be used to control the LPCR via Bluetooth and USB interfaces. The whole system current

consumption is 1.7mA current draw from a 3.7V 650 mAH Li-ion battery. LPCR system can continuous record 30 days ECG data.

Last but not least, to further improve the power performance of previous two wearable ECG systems, a new design of low-power voltage-scalable asynchronous 8051 microcontroller which consumes 40 μW at 1.0V supply is presented in this work. Integrating this asynchronous microcontroller with wearable ECG system may be one area of future work.

Bibliography

[1] http://www.americanheart.org/downloadable/heart/1200082005246HS_Stats%202008.final.pdf

[2] A.S. Go, E.M. Hylek, K.A. Phillips, et al., “Prevalence of diagnosed atrial fibrillation in adults. National implications for rhythm management and stroke prevention: the AnTicoagulation and Risk Factors In Atrial Fibrillation (ATRIA) Study,” *JAMA.*, 285, pp. 2370-2375, 2001.

[3] A. Schuchert, R. Maas, C. Kretzschmar, G. Behrens, I. Kratzman, T. Meinertz, “Diagnostic yield of external electrocardiographic loop recorders in patients with recurrent syncope and negative Tilt table test,” *PACE*, 26, pp.1837-1840, 2003.

[4] Da Ren Zhang; Deepu, C.J.; Xiao Yuan Xu; Yong Lian; , "A wireless ecg plaster for real-time cardiac health monitoring in body sensor networks," *Biomedical Circuits and Systems Conference (BioCAS), 2011 IEEE* , vol., no., pp.205-208, 10-12 Nov. 2011

[5] *Electrocardiography*. Wikipedia, the Free Encyclopedia. 17 Dec. 2009. 02 Jan. 2010. <http://en.wikipedia.org/wiki/Electrocardiography>

[6] Gnecchi, J.A.G.; Vargas, F.O.; Peregrino, V.H.O.; Espinoza, D.L.; , "Design and Construction of a Continuous Ambulatory Electrocardiogram Recorder, Auxiliary in the Detection of Cardiac Arrhythmias," *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010* , vol., no., pp.602-606, Sept. 28 2010-Oct. 1 2010

[7] I-Jan Wang; Lun-De Liao; Yu-Te Wang; Chi-Yu Chen; Bor-Shyh Lin; Shao-Wei Lu; Chin-Teng Lin; , "A Wearable Mobile Electrocardiogram measurement device with novel dry polymer-based electrodes," *TENCON 2010 - 2010 IEEE Region 10 Conference* , vol., no., pp.379-384, 21-24 Nov. 2010 doi: 10.1109/TENCON.2010.5686658

[8] J.Sparsø and S.Furber (eds), *Principles of asynchronous circuit design – A system perspecticve*, Kluwer Academic Publishers, 2001 (ISBN 0-7923-7613-7).

[9] D. Adwards, A. Bardsley, L. Janin and W. Toms, Balsa: *A Tutorial Guide*, version 3.4.2, Jan 2005.

[10] X.D. Zou, X.Y. Xu, L.B. Yao, and Y. Lian, “A 1-V 450-nW Fully Integrated Programmable Biomedical Sensor Interface Chip,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1067-1077, Apr. 2009.

[11] C.J. Deepu, X.Y. Xu, X.D. Zou, L.B. Yao, and Y. Lian, "An ECG-on-Chip for Wearable Cardiac Monitoring Devices" *Fifth IEEE International Symposium on Electronic Design, Test and Application, 2010,* pp 225 – 228, Jan. 2010.

[12] F. Zhang, and Y. Lian, "QRS Detection Based on Multi-Scale Mathematical

Morphology for Wearable ECG Device in Body Area Networks," *IEEE Transactions on Biomedical Circuits and Systems,* Vol.3, No.4, pp.220-228, Aug. 2009.

[13] Microchip,PIC18F46J50 Datasheet, USA

http:// ww1.microchip.com/downloads/en/devicedoc/39931b.

[14] NUS VLSI Lab, V7 Datasheet, Singapore

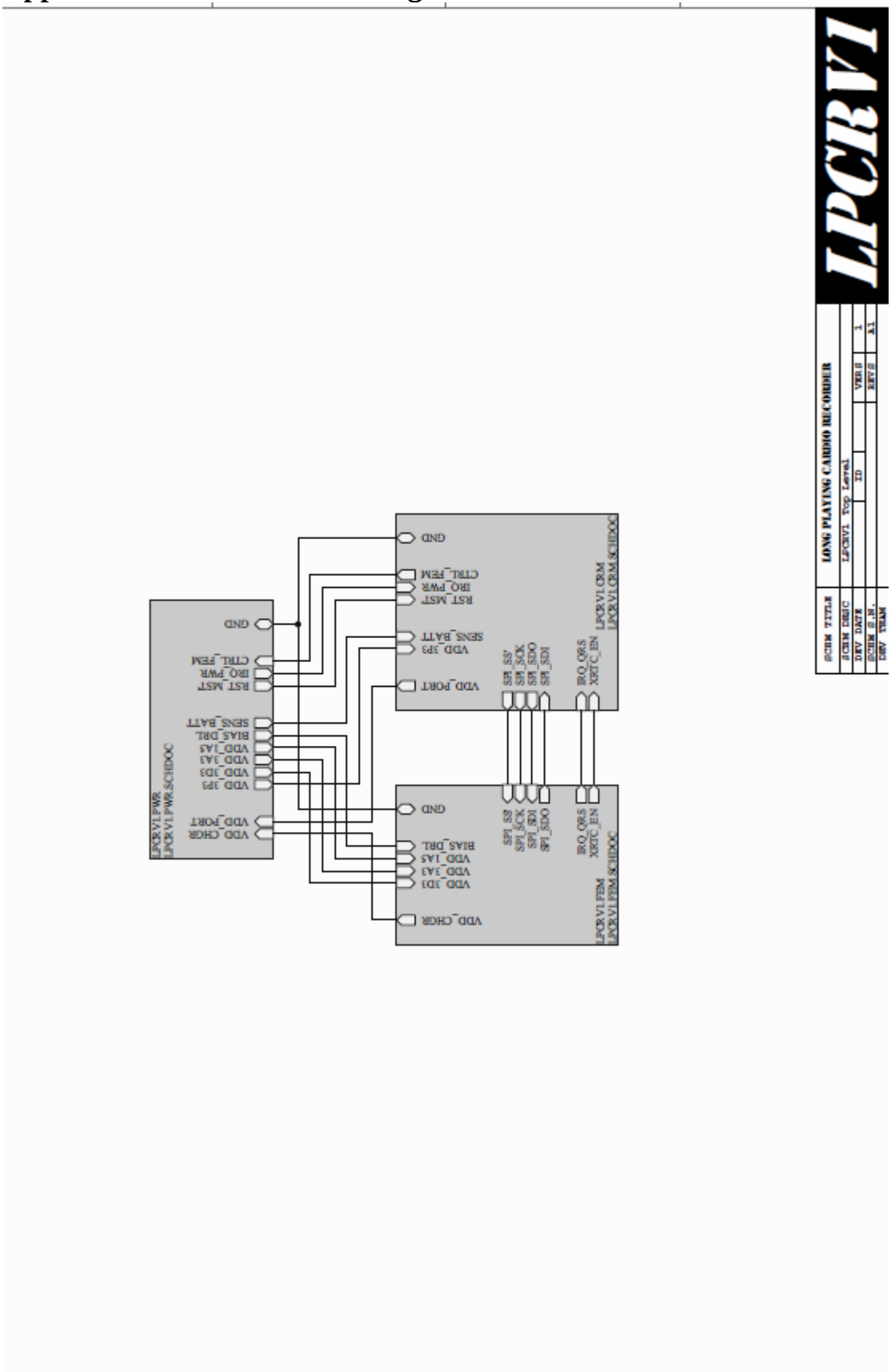
[15] Micron , MT29F16G08FAA Datasheet

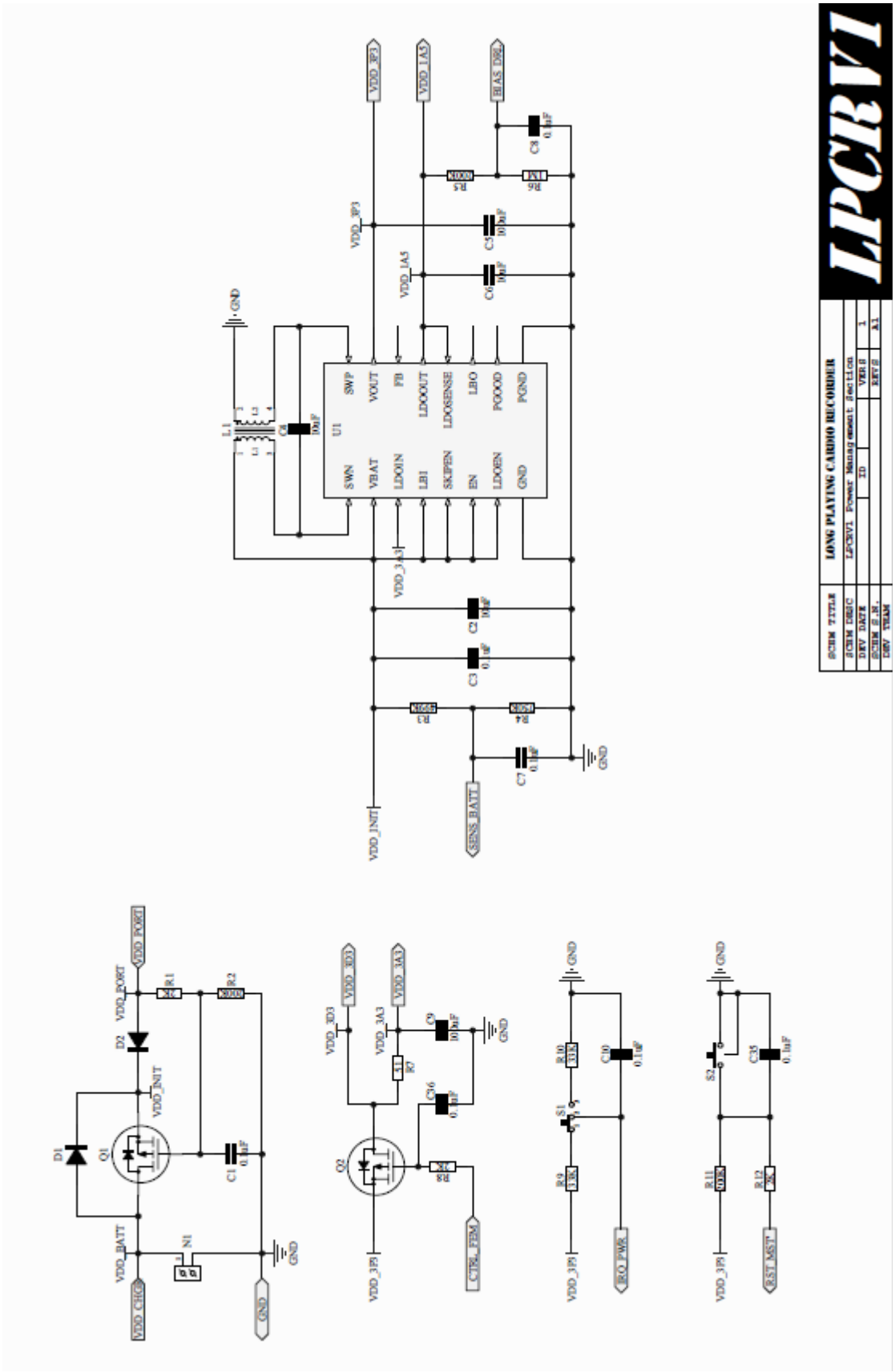
[16] Chao Xue, Xiang Cheng, Yang Guo, Yong Lian, "The Design of a Sub-Nanojoule Asynchronous 8051 with Interface to External Commercial Memory", ASICON, p427-430, 2009.

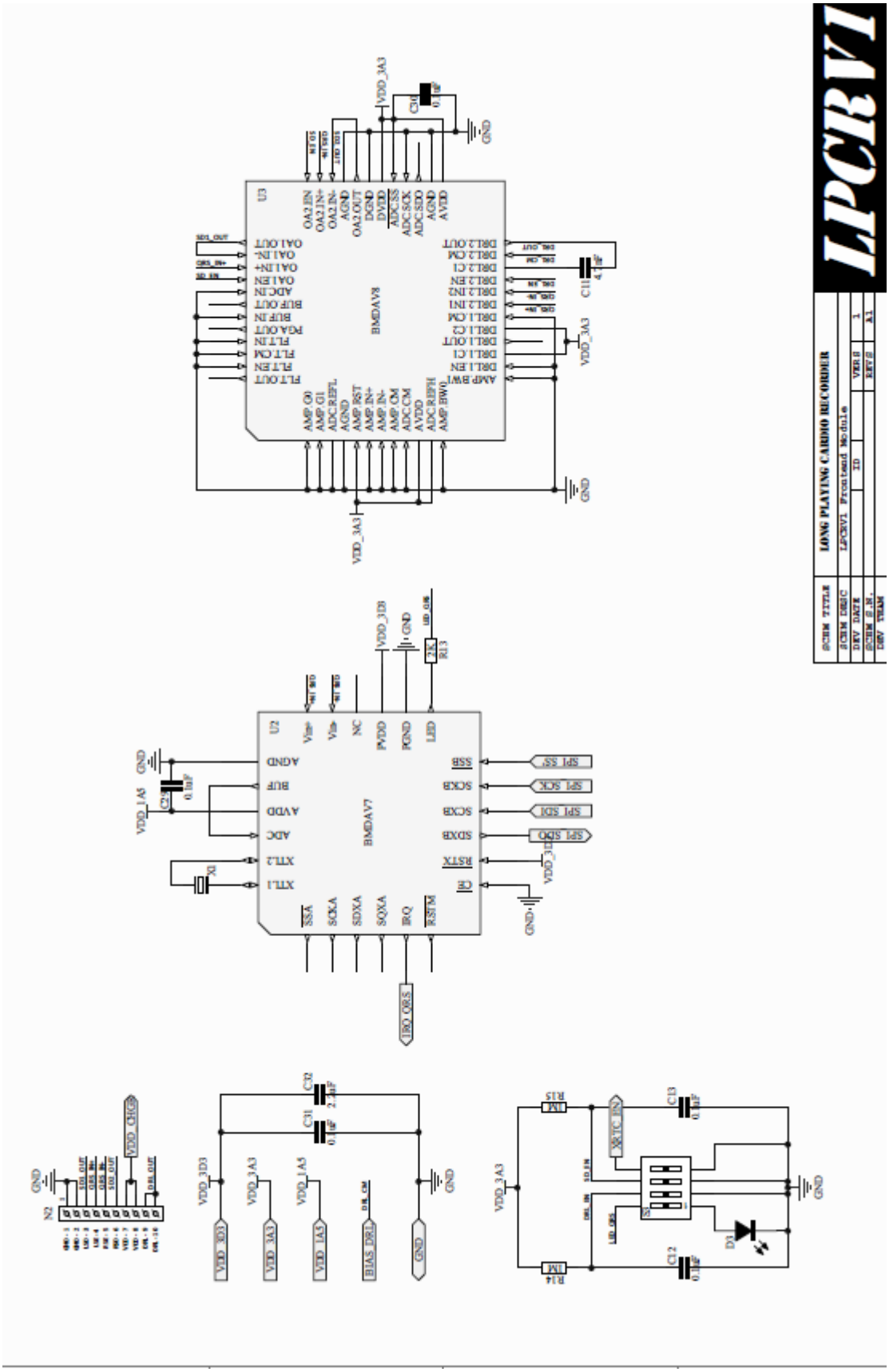
[17] H.V. Gageldonk, K.V. Berkel, A. Peeters, "An Asynchronous low-power 80C51 microcontroller," in Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 30 March-2 April 1998, pp. 96-107

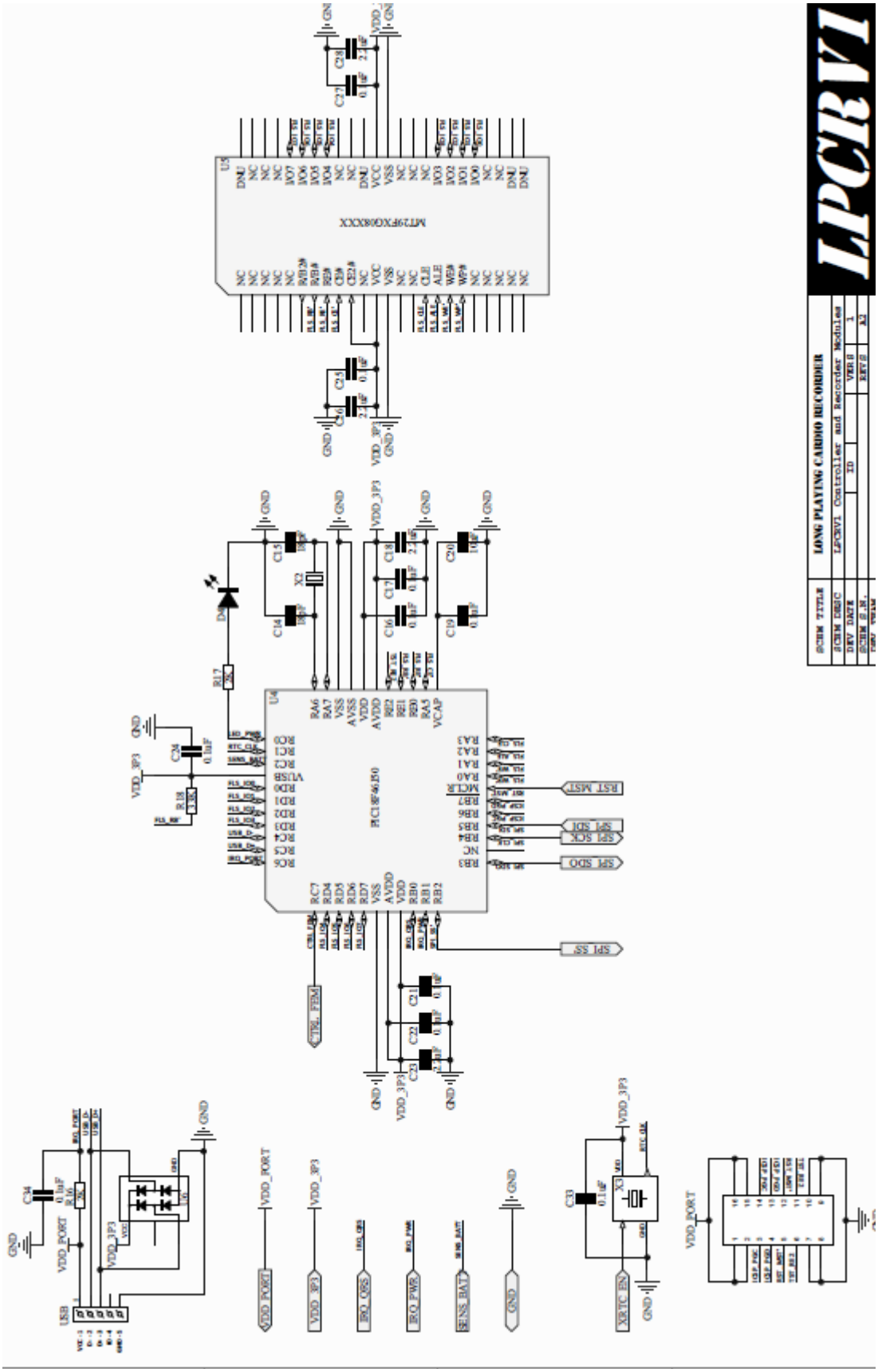
[18] Kok-Leong Chang and Bah-Hwee Gwee, "A low-energy low-voltage asynchronous 8051 microcontroller core" in *International Symposium on Circuits and Systems,* pp. 3181-3184, 2006.

Appendix 1: LPCRV1 PCB design




LPCRVI
LONG PLAYING CARDIO RECORDER
LPCRV1 Power Management Section
VDD_CHG
VDD_BATT
VDD_PORT
VDD_INIT
VDD_3P3
VDD_1A5
BIAS_DRV
SENS_BATT
CTRL_FBM
BIO_PWR
RST_MST
GND
SWN
VBAT
LDOIN
LBI
SKIPEN
EN
LDOEN
GND
SWP
VOUT
FB
LDOOUT
LDOSENSE
LBO
POGOD
PGND
U1
L1
Q1
Q2
D1
D2
N1
S1
S2
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C16
C35


LPCRVI
LONG PLAYING CARDIO RECORDER
LPCRV1 Prototand Module
BMDAV8
BMDAV7
U3
U2
U1
VDD_3A3
VDD_3D3
VDD_1A5
GND
IRQ_ORS
SPI_SS
SPI_SCK
SPI_SDI
SPI_SDO
XRTC_IN
BIAS_DRL
VDD_CHGP
C31
C32
C12
C13
C11
C16
C2
R13
R14
R15
D1
X1
N2


LPCRVI
LONG PLAYING CARDIO RECORDER
LPCRVI Controller and Recorder Modules
MT29FXG08XXXX
PIC18F46J50
U3
U4
U5
X2
X3
D3
R17
R18
R16
C24
C34
C18
C19
C20
C21
C22
C23
C25
C26
C27
C28
C13
C14
C15
C16
C17
C3
VDD_PORT
VDD_3P3
GND
RST_MST
SPI_SDI
SPI_SCK
SPI_SDO
SPI_SS
CTRL_PEM
IRQ_ORS
IRQ_PWR
SENS_BATT
XRTC_EN
RTC_CLK
USB3

Appendix 2: Firmware Flash part

```
#include "p18cxxx.h"
#include "ee5003conf.h"
#include "funflash.h"
#include "myrtcc.h"


extern void rdputchar(unsigned char ucSd);
extern void putcharhex(unsigned char ucSd);
extern void myprint_rom(rom const char *pStr);
extern void myprint_uint32HEX(uint32 u32Data);
extern void putcharhex(unsigned char ucSd);

#define FLASH_READ_READY()  while(FLASH_READYBUSY_L_PORT_IN==0)

//Low power mode
void flash_disabled(void)
{
    FLSH_CE1_L_HIGH();
}

/*
Set the read-mode command input
*/
static void __flash_readmode_setcommand(uint8 u8Cmd)
{
    FLSH_CLE_HIGH();
    FLSH_ALE_LOW();
    FLSH_CE1_L_LOW();
    FLSH_RE_L_HIGH();
    FLSH_WE_L_LOW();
    PIC_DATA_BUS_SET_AS_OUTPUT(); /*IO Direction as Output*/
    FLASH_DATABUS_PORT_OUT=u8Cmd;
    Nop();
    FLSH_WE_L_HIGH();
}

/*
Set the write-mode command input
*/
static void __flash_writemode_setcommand(uint8 u8Cmd)
{
    FLSH_WP_L_HIGH();
    __flash_readmode_setcommand(u8Cmd);
}

/*
Set the read-mode address input
*/
static void __flash_readmode_setaddress(uint8 u8Addr8Bits)
{
    FLSH_CLE_LOW();
    FLSH_ALE_HIGH();
    FLSH_CE1_L_LOW();
    FLSH_RE_L_HIGH();
    FLSH_WE_L_LOW();
    PIC_DATA_BUS_SET_AS_OUTPUT(); /*IO Direction as Output*/
    FLASH_DATABUS_PORT_OUT=u8Addr8Bits;
    Nop();
    FLSH_WE_L_HIGH();
}

/*
Set the write-mode address input
*/
static void __flash_writemode_setaddress(uint8 u8Addr8Bits)
{
    FLSH_WP_L_HIGH();
    __flash_readmode_setaddress(u8Addr8Bits);
}
```

```
/*Write data*/
static void __flash_writedata(uint8 u8Data)
{
    FLSH_WP_L_HIGH();
    FLSH_CLE_LOW();
    FLSH_ALE_LOW();
    FLSH_CE1_L_LOW();
    FLSH_RE_L_HIGH();
    FLSH_WE_L_LOW();
    PIC_DATA_BUS_SET_AS_OUTPUT(); /*IO Direction as Output*/
    FLASH_DATABUS_PORT_OUT=u8Data;
    Nop();
    FLSH_WE_L_HIGH();
}

/*Read Data*/
static uint8 __flash_readdata(void)
{
    uint8 uTmpData;

    FLSH_CLE_LOW();
    FLSH_ALE_LOW();
    FLSH_CE1_L_LOW();
    FLSH_RE_L_HIGH();
    FLSH_WE_L_HIGH();
    PIC_DATA_BUS_SET_AS_INPUT(); /*IO Direction as Input*/
    while(FLASH_READYBUSY_L_PORT_IN==0);
    FLSH_RE_L_LOW();
    Nop(); /*Delay*/
    Nop(); /*Delay*/
    Nop(); /*Delay*/
    uTmpData=FLASH_DATABUS_PORT_IN;
    FLSH_RE_L_HIGH();
    return uTmpData;
}

void hw_flash_init(void)
{
    FLASH_WE_TRIS_L=0; /*Write Enable;  PIC's Output*/
   FLASH_WP_TRIS_L=0; /*Write Protect; PIC's Output*/
   FLASH_CLE_TRIS=0; /*PIC's Output*/
   FLASH_ALE_TRIS=0;
   FLASH_CE1_TRIS_L=0;
   FLASH_RE_TRIS_L=0;
   FLASH_READYBUSY_L_TRIS_IN=1; /*Input. Ready=1, Busy=0*/
   /*initialize port*/
   FLASH_WE_PORT_L=1;
   FLASH_WP_PORT_L=1;
   FLASH_CLE_PORT=0;
   FLASH_ALE_PORT=0;
   FLASH_CE1_PORT_L=1;
   FLASH_RE_PORT_L=1;
}
//
//void _set_command_readmode(uint8 uData)
//{
//    FLASH_CLE_PORT=1;
//    FLASH_ALE_PORT=0;
//    FLASH_CE1_PORT_L=0;
//    FLASH_RE_PORT_L=1;
//    FLASH_WP_PORT_L=0;
//    FLASH_WRITE_PORT_L=?;
//    FLASH_WRITE_PORT_L=1;
//    FLASH_WRITE_PORT_L=0;
//}
//
```

```
/*
Pre-request:
1. Data bus is Input : call PIC_DATA_BUS_SET_AS_INPUT();
2. FLSH_CLE_LOW();  The CLE must be low
*/
uint8 _flash_read_one_byte(void)
{
    uint8 uTmp;
    //while(FLASH_READYBUSY_L_PORT_IN==0);
    FLASH_READ_READY();
    FLSH_RE_L_LOW();
    Nop(); /*Delay*/
    uTmp=FLASH_DATABUS_PORT_IN;
    FLSH_RE_L_HIGH();
//putcharhex(uTmp);
    return uTmp;
}

uint32 myFlashGetID(void)
{
    uint32 u32ID;
    __flash_readmode_setcommand(0x90);
    __flash_readmode_setaddress(0x00);

    u32ID=0;
    u32ID|=__flash_readdata();
    u32ID<<=8;
    u32ID|=__flash_readdata();
    u32ID<<=8;
    u32ID|=__flash_readdata();
    u32ID<<=8;
    u32ID|=__flash_readdata();
    return u32ID;
}
uint32 oldmyFlashGetID(void)
{
    uint32 u32ID;
    PIC_DATA_BUS_SET_AS_OUTPUT(); /*IO Direction as Output*/
    FLSH_CLE_HIGH();
    FLSH_CE1_L_LOW();
    FLSH_WE_L_LOW();
    FLSH_ALE_LOW();
    FLSH_RE_L_HIGH();
    FLASH_DATABUS_PORT_OUT=CMD_READ_ID;
    FLSH_WE_L_HIGH();
    FLSH_CLE_LOW();

    FLSH_ALE_HIGH();
    FLSH_WE_L_LOW();
    FLASH_DATABUS_PORT_OUT=0x00; /*Address, 1-cycle*/
    FLSH_WE_L_HIGH();
    FLSH_ALE_LOW();

    PIC_DATA_BUS_SET_AS_INPUT(); /*IO Direction as Input*/
    u32ID=0;
    u32ID|=_flash_read_one_byte();
    u32ID<<=8;
    u32ID|=_flash_read_one_byte();
    u32ID<<=8;
    u32ID|=_flash_read_one_byte();
    u32ID<<=8;
    u32ID|=_flash_read_one_byte();
    return u32ID;
```

```
}

/************Basic Functions***************/
/*
This function just initialize the Read-Page feature.
Parameter(s):
u16CAaddr: 16-bit column address (first and second cycle)
    Value: address value should be <(2048+64)
u32PageBlockAddress: 32-bit block+page address (third to fifth cycle)
*/
void myflash_read_page_init(uint16 u16CAaddr, uint32 u32PageBlockAddress)
{
    uint8 uTmpAddr;
    uint16 uTmpColAddr=u16CAaddr;
    __flash_readmode_setcommand(0x00);
    /*First: column address CA0~CA7*/
    uTmpAddr=uTmpColAddr & 0xFF;
    uTmpColAddr>>=8;
    __flash_readmode_setaddress(uTmpAddr);
    /*Second: column address CA8~CA11*/
    uTmpAddr=uTmpColAddr & 0xFF;
    __flash_readmode_setaddress(uTmpAddr);

    /*Third: Page address (PA0~PA6) + Block address (BA6~BA7)*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_readmode_setaddress(uTmpAddr);
    /*4th: Block address BA8~BA15*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_readmode_setaddress(uTmpAddr);
    /*5th: Block address BA16~BA18*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    __flash_readmode_setaddress(uTmpAddr);

    /*Set the command, 0x30*/
    __flash_readmode_setcommand(0x30);

    FLASH_READ_READY();
    PIC_DATA_BUS_SET_AS_INPUT();
    FLSH_CLE_LOW();

//    /*Following set the random data read*/ //ignored,
//    __flash_readmode_setcommand(0x05);
//    uTmpColAddr==u16CAaddr;
//    /*First: column address CA0~CA7*/
//    uTmpAddr=uTmpColAddr & 0xFF;
//    uTmpColAddr>>=8;
//    __flash_readmode_setaddress(uTmpAddr);
//    /*Second: column address CA8~CA11*/
//    uTmpAddr=uTmpColAddr & 0xFF;
//    __flash_readmode_setaddress(uTmpAddr);
//
//    __flash_readmode_setcommand(0xE0);
}

uint8 myflash_read_one_byte()
{
    return _flash_read_one_byte();
}


/*
```

```
This function uses "Program Page Cache Mode" to program data.
Maximum input length is 255
Parameter(s):
    u16CAaddr: 16-bit column address (first and second cycle)
        Value: address value should be <(2048+64)
    u32PageBlockAddress: 32-bit block+page address (third to fifth cycle)
    pDataSrc: buffer pointer in the memory
    u16Len: length of the data
return:
    0: Successfully
    any other values: Status Value, which is not zero
*/
uint8 myfalsh_write_page_string(uint16 u16CAaddr, uint32 u32PageBlockAddress, uint8 *pDataSrc, uint16
u16Len)
{
    uint16 u16Count;
    uint8 u8Status;
    uint8 uTmpAddr;
    uint16 uTmpColAddr=u16CAaddr;
    __flash_writemode_setcommand(0x80);
    /*First: column address CA0~CA7*/
    uTmpAddr=uTmpColAddr & 0xFF;
    uTmpColAddr>>=8;
    __flash_writemode_setaddress(uTmpAddr);
    /*Second: column address CA8~CA11*/
    uTmpAddr=uTmpColAddr & 0xFF;
    __flash_writemode_setaddress(uTmpAddr);

    /*Third: Page address (PA0~PA6) + Block address (BA6~BA7)*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_writemode_setaddress(uTmpAddr);
    /*4th: Block address BA8~BA15*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_writemode_setaddress(uTmpAddr);
    /*5th: Block address BA16~BA18*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    __flash_writemode_setaddress(uTmpAddr);

    /*Preparing writing data*/
    for(u16Count=0;u16Count<u16Len;u16Count++)
    {
        __flash_writedata(pDataSrc[u16Count]);
    }
    /*Set the command, 0x15*/
    __flash_writemode_setcommand(0x15);

    FLASH_READ_READY();
    __flash_readmode_setcommand(0x70);

    u8Status=myflash_read_one_byte();
    if(u8Status&0x3)
    {/*Failed*/
        return u8Status;
    }
    return (0);/*Successfully*/
}

/*
FunctionName: myflash_erase_block
    Requires page+block address. however, the page address is ignored.
Parameter:
```

```
    u32PageBlockAddress: 32-bit block+page address (third to fifth cycle)
                         here, the page address is ignored.
return:
    0: Successfully
    others: failed (status register value)
*/
uint8 myflash_erase_block(uint32 u32PageBlockAddress)
{
    uint8 uTmpAddr,u8Status;
    __flash_writemode_setcommand(0x60);

    /*Third: Page address (PA0~PA6) + Block address (BA6~BA7)*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_writemode_setaddress(uTmpAddr);
    /*4th: Block address BA8~BA15*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    u32PageBlockAddress>>=8;
    __flash_writemode_setaddress(uTmpAddr);
    /*5th: Block address BA16~BA18*/
    uTmpAddr=u32PageBlockAddress & 0xFF;
    __flash_writemode_setaddress(uTmpAddr);

    __flash_writemode_setcommand(0xD0);

    FLASH_READ_READY();
    __flash_readmode_setcommand(0x70);

    for(uTmpAddr=0;uTmpAddr<250;uTmpAddr++)
    {/*Check complete or not*/
        u8Status=myflash_read_one_byte();
        if(u8Status&0x40) /*bit 6, 1 means complete*/
        {/*Completed*/
            if(u8Status & 0x1)
            {/*error*/
                return u8Status;
            }
            else /*bit0=0*/
            {/*successfully*/
                return 0;
            }
        }
    }
    return (u8Status);/*Timeout*/
}


/***************************High Level Functions*******************************/
/*
FunctionName:
    myflash_check_page_blank
Parameter:
    u32PageBlockAddress: 32-bit block+page address (third to fifth cycle)
                         here, the page address is ignored.
Return:
    the index of writable byte-region.
    e.g. 0: means this page is blank
            1: means the first byte (0,1,...2112) (started from index=0) is blank, it is available.
    0xFFFF means: page is full used (including last 64 bytes).
*/
uint16 myflash_check_page_blank(uint32 u32PageBlockAddress)
{
    uint16 u16ColumAddress=0;
```

```
    uint16 i;
    uint8 u8Tmp1,u8FoundNoneZero=0;
    myflash_read_page_init(u16ColumAddress,u32PageBlockAddress);

    for(i=0;i<SIZE_ONE_PAGE;i++)
    {
        u8Tmp1=myflash_read_one_byte();
        if(u8Tmp1!=0xFF)
        {
            u8FoundNoneZero=1;
            u16ColumAddress=i; /*update the u16ColumnAddress*/
        }
    }
    if(u8FoundNoneZero)
    {
        if(u16ColumAddress>=SIZE_ONE_PAGE)
        {
            return 0xFFFF; /*error!*/
        }
        return u16ColumAddress+1;
    }
    return 0;/*u16ColumAddress; 0 index, page is blank*/
}

//uint8 myflash_set_page_using_used(uint32 u32PageBlockAddress)

///*
//Function: myflash_get_last_using_block_index
//    To get the block, which is not full.
//    Returned value is the block index.
//Return: 0: failed
//        others, OK.
//*/
//uint16 myflash_get_last_using_block_index(void)
//{
//    uint16 uTmpBlockAddress=0;
//    uint16 i;
//    uint8 u8Tmp1;
//    stFlashBlockInformation sBlkInfo;
//    myflash_read_page_init(0, 0); /*initialize the read page command: try to read page 0*/
//    for(i=1;i<SIZE_ONE_PAGE;i++) /*started from 1 to 2047*/
//    {
//        u8Tmp1=myflash_read_one_byte();
//        sBlkInfo.blkinfo.uBlkInfoData=u8Tmp1;
//
//        if(sBlkInfo.blkinfo.bits.bBlockIsFull==1)
//        {/*bBlockIsFull=0: Full; =1:Not Full*/
//            uTmpBlockAddress=i; /*update the u16ColumnAddress*/
//            return uTmpBlockAddress;
//        }
//    }
//    return 0;
//}

///*
//Function Name:
//    To update and mark the specified block index as full.
//Parameter:
//    u16BlockIndex: give the block index, value range is 0~2047
//return:
//    0: Successfully
//    others: failed (status register value)
//*/
```

```
//uint8 myflash_mark_block_as_full(uint16 u16BlockIndex)
//{
//    uint16 u8ByteIndex; /*the byte index in the page 0*/
//    stFlashBlockInformation stBlkData;
//    uint8 uDataBuf[2];
//    stBlkData.blkinfo.uBlkInfoData=0xFF;
//    stBlkData.blkinfo.bits.bBlockIsFull=0; /*Set bit as low*/
//    uDataBuf[0]=stBlkData.blkinfo.uBlkInfoData;
//    uDataBuf[1]=0;
//    return myfalsh_write_page_string(u16BlockIndex, 0, uDataBuf, 2);
//}

///*
//Function Name:
//    To update and mark the specified block index as using.
//Parameter:
//    u16BlockIndex: give the block index, value range is 0~2047
//return:
//    0: Successfully
//    others: failed (status register value)
//*/
//uint8 myflash_mark_block_as_using(uint16 u16BlockIndex)
//{
//    uint16 u8ByteIndex; /*the byte index in the page 0*/
//    stFlashBlockInformation stBlkData;
//    uint8 uDataBuf[2];
//    stBlkData.blkinfo.uBlkInfoData=0xFF;
//    stBlkData.blkinfo.bits.bBlockIsUsing=0; /*Set bit as low*/
//    uDataBuf[0]=stBlkData.blkinfo.uBlkInfoData;
//    uDataBuf[1]=0;
//    return myfalsh_write_page_string(u16BlockIndex, 0, uDataBuf, 2);
//}
//

///*
//Function:
//    With the given block index, try to find the first blank page, which is not using.
//    Information is stored to the last 64-byte.
//    (2048,2049: reserved)
//Return:
//    0~63 : Valid page index number
//    0xFFFF: all pages are full.
//*/
//uint16 myflash_get_block_first_blank_page_index(uint16 u16BlockIndex)
//{
//    uint8 i=0;
//    uint8 u8Data;
//    uint32 u32TmpPageBlockAddr=0;
//    u32TmpPageBlockAddr=u16BlockIndex;
//    u32TmpPageBlockAddr <<= 6; /*Convert it from block index to page-block address*/
//
//    for(i=0;i<PAGES_PER_BLOCK;i++)
//    {
//        myflash_read_page_init(0, u32TmpPageBlockAddr);
//        u32TmpPageBlockAddr++;
//        u8Data=myflash_read_one_byte();
//        if(u8Data==0xFF)
//        {/*this page is not used, return this address*/
//            return i;
//        }
//    }
//    return 0xFFFF;
//}
```

```
//
/*
Slow speed operation.
Parameter(s):
    u16CAaddr: 16-bit column address (first and second cycle)
        Value: address value should be <(2048+64)
    u32PageBlockAddress: 32-bit block+page address (third to fifth cycle)
    u8Data: one but to be write to address
return:
    0: Successfully
    any other values: Status Value, which is not zero
*/
uint8 myWriteOneByte(uint16 u16CAaddr, uint32 u32PageBlockAddress, uint8 u8Data)
{
    uint8 uTmpBuf2[2];
    uTmpBuf2[0]=u8Data;
    uTmpBuf2[1]=0;
    return myfalsh_write_page_string(u16CAaddr,u32PageBlockAddress,uTmpBuf2,1);
}

/*
Slow speed operation.
Returned Value:
    read data.
*/
uint8 myReadOneByte(uint16 u16CAaddr, uint32 u32PageBlockAddress)
{
    myflash_read_page_init(u16CAaddr, u32PageBlockAddress);
    return myflash_read_one_byte();
}

/*Simple File System Functions*/
/*
Each block's first page includes the page usage information.
Its last 64byte (in the padding area) shows such informatino.
Return:
0: Page is Blank
1: Page is Using (end of previous record)
2: Page is Used
3: Page is Damaged /try to erase this block, and test again, if still failed, skip this block (unknown
status)
*/
uint8 mySFSCheckPageBlank(uint32 u32PageBlockAddress)
{
    uint16 uTmpCAaddr;
    uint8 u8Ret;
    uTmpCAaddr=2048+(u32PageBlockAddress)&0x3F;
    u8Ret=myReadOneByte(uTmpCAaddr,u32PageBlockAddress);
    switch(u8Ret)
    {
        case FLSH_SFS_PAGE_BLANK:
            return 0;
        case FLSH_SFS_PAGE_USING:
            return 1;
        case FLSH_SFS_PAGE_USED:
            return 2;
            break;
        default:
            return 3;
    }
}

/*
```

```
Return:
0: Block is Blank
1: Block is partially used. (Not Bland)
2: Block is all used. (Not Blank)
*/
uint8 mySFSCheckBlockBlank(uint32 u32PageBlockAddress)
{
    uint16 uTmpCAaddr=2048;
    uint8 u8Ret,i,iRet=0,iUsedPagesCount=0;
    myflash_read_page_init(uTmpCAaddr,(u32PageBlockAddress & 0xFFFFFFC0));
    for(i=0;i<PAGES_PER_BLOCK;i++)
    {
        u8Ret=myflash_read_one_byte();
        if(u8Ret!=0xFF)
        {
            //putcharhex(u8Ret);
            //putchar(' ');

            iUsedPagesCount++;
        }
    }
    if(iUsedPagesCount==PAGES_PER_BLOCK)
    {
        iRet=2; //fully used
    }
    else if(iUsedPagesCount >0 )
    {
        iRet=1; //partially used
    }

    return iRet;
}

uint8 mySFSGetAvailablePageIndexInThisBlock(uint8 *pValidPageIndex, uint32 u32PageBlockAddress)
{
    uint16 uTmpCAaddr=2048;
    uint8 u8Ret,i;
    myflash_read_page_init(uTmpCAaddr,(u32PageBlockAddress & 0xFFFFFFC0));

    *pValidPageIndex=0;
    for(i=0;i<PAGES_PER_BLOCK;i++)
    {
        u8Ret=myflash_read_one_byte();
        if(u8Ret==0xFF)
        {
            *pValidPageIndex=i;
            return 0;
        }
    }
    return 1; //failed
}

/*
This function should only be called once. (During system start-up).
Return:
0: OK
1: Failed, checked all the pages, all full (need erasing the first block. and continue...
*/
uint8 mySFSGetAvailablePageBlockAddress(uint32 *pRetU32PageAddress)
{
    uint32 uPA=0;
    const uint32 uMax=524224;//8G is 524224(8191*64)  131072;
    const uint32 uStep=64;
```

```
    uint8 u8Ret=0xff,u8PageIndex=0;
myprint_rom("\r\nmySFSGetAvailablePageBlockAddress");//\r\n");
//myprint_uint32HEX(uPA);
myprint_rom("\r\nMaxPageAddr=");
myprint_uint32HEX(uMax);
myprint_rom("\r\nStep=");
myprint_uint32HEX(uStep);
myprint_rom("\r\n");

    for(;uPA<uMax;uPA+=uStep)
    {
        u8Ret=mySFSCheckBlockBlank(uPA);
        if(u8Ret==0)
        {//This block is available
            myprint_rom("\r\nFound Page-Block Address:");
            myprint_uint32HEX(uPA);
            myprint_rom("\r\n");
            *pRetU32PageAddress=uPA;
            return 0; //OK
        }
        else if(u8Ret==1) //This block is partially used
        {//continue get the address
            u8Ret=mySFSGetAvailablePageIndexInThisBlock(&u8PageIndex,uPA);
            if(u8Ret==0)
            {//got it
                myprint_rom("\r\nFound Page-Block Address:");
                myprint_uint32HEX(uPA);
                myprint_rom("\r\n");
                *pRetU32PageAddress=uPA+u8PageIndex; //available page-block address
                return 0;
            }
        }
        else
        {//all blocks' pages are used, move to next block
            //do nothing.
        }
    }
myprint_rom("\r\nFun:Failed,All Pages Full\r\n");
    return 1; //failed, checked all the pages, all full
}


void mySFSUpdatePageIsUsing(uint32 u32PageBlockAddress)
{
    uint16 uCA=u32PageBlockAddress & 0x3F ; //get the page address
    uCA+=2048; //refer to the page's info.
    u32PageBlockAddress &= 0xFFFFFFC0;
    myWriteOneByte(uCA,u32PageBlockAddress,FLSH_SFS_PAGE_USING);
}

void mySFSUpdatePageUsed(uint32 u32PageBlockAddress)
{
    uint16 uCA=u32PageBlockAddress & 0x3F ; //get the page address
    uCA+=2048; //refer to the page's info.
    u32PageBlockAddress &= 0xFFFFFFC0;
    myWriteOneByte(uCA,u32PageBlockAddress,FLSH_SFS_PAGE_USED);
}

/*
To update the string to flash.
Flash next address will also be updated.
1. Check the next block available? If not, erase it. Check again, if still not available, skip and move
to the next one.
```

```
   Do the same check, if has data, format, and check again. and etc.
2. Update the SFS status in each block's page 0.
3. Before update the last's page status to SFS, always erase the next block.)
4. Update Using/Used for SFS.
*/
uint8 mySFSUpdateDataToFlash(uint8 *pDataSrc, uint16 u16Len, uint16 *p16CAaddr, uint32
*p32PageBlockAddress)
{
    uint32 u32TmpAddr;
    uint8 uTmpPageIndex;
    uint16 uTmpLen;
    uint8 uDateTimeDemo[8];//={0x20,0x11,0x01,0x16,0x23,0x45,0x22,0x00,0x00};

    uDateTimeDemo[0]=0x20;
    rtc_getdatetime(&uDateTimeDemo[1],&uDateTimeDemo[2],&uDateTimeDemo[3],
                    &uDateTimeDemo[4],&uDateTimeDemo[5],&uDateTimeDemo[6]);

#ifdef DEBUG_MSG
    myprint_rom("\r\nLen=");
    putcharhex(u16Len>>8);
    putcharhex(u16Len&0xFF);
    myprint_rom("; CA=");
    putcharhex((*p16CAaddr)>>8);
    putcharhex((*p16CAaddr)&0xFF);
    myprint_rom("; PageBlock=");
    myprint_uint32HEX(*p32PageBlockAddress);
    myprint_rom("\r\n");
#endif

    if (*p16CAaddr ==0)
    {
        mySFSUpdatePageIsUsing(*p32PageBlockAddress);
    }

    if (*p16CAaddr >= 2048)
    {//
//putchar('A');
        mySFSUpdatePageUsed(*p32PageBlockAddress);
        *p16CAaddr=0;
        (*p32PageBlockAddress)+=1; //page increase by one
        mySFSUpdatePageIsUsing(*p32PageBlockAddress);
        uTmpPageIndex=(*p32PageBlockAddress)&0x3F;
        u32TmpAddr=(*p32PageBlockAddress) + 32; //move to next block

        if(uTmpPageIndex ==63)
        {//erasing next block
            if(u32TmpAddr >= (524224))
            {//end of 8Gb, move to block 0
                u32TmpAddr=0;
            }
            myflash_erase_block(u32TmpAddr);
//putchar('B');
        }
    }

    while(u16Len>0)
    {
//putchar('C');
        if(*p16CAaddr ==0)
        {//Write date,time, seconds
            myfalsh_write_page_string(*p16CAaddr,*p32PageBlockAddress,uDateTimeDemo,6); //write
temperary date/time
            (*p16CAaddr)+=6;
```

```
//putchar('D');
        }

        uTmpLen=2048-(*p16CAaddr);
        if(uTmpLen > u16Len)
        {
//putchar('E');
            uTmpLen=u16Len;
        }

        if((*p32PageBlockAddress)>=(524224))
        {
            *p32PageBlockAddress=0; //move back to block 0
        }

        myfalsh_write_page_string(*p16CAaddr,*p32PageBlockAddress,pDataSrc,uTmpLen);
        u16Len-=uTmpLen;
        *p16CAaddr+=uTmpLen;
        pDataSrc+=uTmpLen;

        if((*p16CAaddr)>=2048)
        {
//putchar('G');
            mySFSUpdatePageUsed(*p32PageBlockAddress);
            *p16CAaddr=0;
            (*p32PageBlockAddress)+=1; //page increase by one
            mySFSUpdatePageIsUsing(*p32PageBlockAddress);
            uTmpPageIndex=(*p32PageBlockAddress)&0x3F;
            u32TmpAddr=(*p32PageBlockAddress) + 32; //move to next block

            if(uTmpPageIndex ==63)
            {//erasing next block
                if(u32TmpAddr >= 524224)
                {//end of 8Gb, move to block 0
                    u32TmpAddr=0;
                }
                myflash_erase_block(u32TmpAddr);
//putchar('H');
            }
        }
    }
}
```

Appendix 3: Asynchronous 8051 core Balsa code

```
import [balsa.types.basic]
import [defines]


procedure Balsa_NOPIPE(
input  rom_data               : byte;
output rom_addr               : Imm16;
output rom_rd          : bit;

--external RAM and ROM
output xram_addr       : Imm16;
output xram_out_data   : byte;
input  xram_in_data    : byte;
output tris_data       : bit;
output xram_rd         : bit;
output xram_wr         : bit;
output xram_e          : bit; --xram access

output int_return      : bit;
input  int_req         : Imm3;
input  int_mask               : Imm2;
output int_lproc       : bit;
output int_lend               : bit;

(--
--timer and iram access
output cpu_out_data    : byte;
input  cpu_in_data     : byte;
output cpu_rd          : bit;
output cpu_wr          : bit;
output cpu_addr               : byte;
output cpu_is_bit_addr : bit;
output cpu_out_bit_data : bit;
input  cpu_in_bit_data  : bit;
--)

--Asyn RAM interface
input  iram_out_data          : byte;
output iram_in_data           : byte;
output iram_addr              : Imm7;
output iram_rd                : bit;
output iram_wr                : bit;

--timer interface
output timer_out_data         : byte;
output timer_out_bit_data     : bit;
output timer_addr             : byte;
output timer_bit_addr         : bit;
output timer_rd                       : bit;
output timer_wr                       : bit;
input  timer_in_data          : byte;
input  timer_in_bit_data      : bit;


--parallel port interface
output P0_o                   : byte;
output P1_o                   : byte;
output P2_o                   : byte;
input  P0_i                   : byte;
input  P1_i                   : byte;
input  P2_i                   : byte;
output tris0_o                : byte;
output tris1_o                : byte;
output tris2_o                : byte;


--testing chans
output pc_out          : Imm16;
output acc_out         : byte;
```

```
output psw_out         : byte;
output debug1 :Imm2;

output sp_out                 : byte;
output dpl_out                : byte;
output dph_out                : byte;
output b_out                  : byte
) is
        variable fetch_op      : Imm2

        variable reg_pc_plus   : Imm16
        variable exe_opcode    : Imm7

        variable data_bit      : bit
        variable reg_pc,reg_pc_tmp    : Imm16
        variable data_bus      : byte
        variable reg_ir        : byte
        variable reg_op1,reg_op2,reg_op3      : byte
        variable reg_acc       : byte
        variable int_req_int : Imm3
        variable int_mask_int : Imm2
        variable int_hproc_int,int_lproc_int : bit

        variable reg_pcl,reg_pch                      : byte
        variable reg_pc_11_15                 : Imm5
        variable reg_pc_8_10                  : Imm3
        variable reg_pc_0_7                   : byte
--      variable cpu_state,exe_state                  : Imm2
        variable reg_f0,reg_cy,reg_ac,reg_ov,reg_rs1,reg_rs0,reg_nu,reg_p : bit

--      variable DEC

        variable op_in_int     : byte
        variable op_out_int    : array7
        variable rmw_int       : bit

--      variable ALU

        variable v1, v16 : Imm16
        variable v2, v : Imm9
        variable alu_des_2, alu_src_1, alu_src_2, alu_src_3 : byte
        variable v4 : Imm5
        variable alu_op_code, v8 : Imm4
        variable vC : Imm2
        variable alu_des_cy, alu_des_ac, alu_des_ov, alu_src_cy, alu_src_ac : bit

        variable alu_des_1 : array8
        variable v4_4, v8_3 : bit

--      variable IRAM
        variable P0,P1,P2,SP,DPL,DPH,ACC,B,PSW : byte
        variable P0_out,P1_out,P2_out,tris0,tris1,tris2      : byte
        variable ram_in_data,ram_out_data,ram_addr,iram_out_data_int : byte
        variable ram_is_bit_addr      : bit
        variable ram_in_bit_data,ram_out_bit_data : bit
        variable temp_data,temp_addr : array8



--      final 1st version
        variable ram_out_data_tmp, ram_addr_tmp, xram_out_data_tmp  : byte
        variable ram_out_bit_data_tmp : bit
        variable reg_psw       : byte
        variable xram_addr_tmp : Imm16



        shared GET_RAM_ADDR_1 is
        begin
            ram_addr_tmp := (#reg_op1[0..2] @ #reg_rs0 @ #reg_rs1 @ {0,0,0} as byte)
        end

        shared GET_RAM_ADDR_2 is
```

```
        begin
            ram_addr_tmp := (#reg_op1[0..0] @ {0,0} @ #reg_rs0 @ #reg_rs1 @ {0,0,0} as
byte)
        end

        shared GET_PSW is
        begin
            PSW:=(#reg_p@#reg_nu@#reg_ov@#reg_rs0@#reg_rs1@#reg_f0@#reg_ac@#reg_cy as
byte)
        end


(--
shared TO_TIMER is
begin
        cpu_addr <- ram_addr || cpu_is_bit_addr <- ram_is_bit_addr || cpu_rd <- ram_rd
|| cpu_wr <- ram_wr
end
--)

shared READ_TIMER is
begin
        timer_addr <- ram_addr || timer_bit_addr <- 0b0 || timer_rd <- 0b1
end

shared READ_BIT_TIMER is
begin
        timer_addr <- ram_addr || timer_bit_addr <- 0b1 || timer_rd <- 0b1
end

shared WR_TIMER is
begin
        timer_addr <- ram_addr || timer_bit_addr <- 0b0 || timer_wr <- 0b1 ||
        timer_out_data <- ram_out_data
end

shared WR_BIT_TIMER is
begin
        timer_addr <- ram_addr || timer_bit_addr <- 0b1 || timer_wr <- 0b1 ||
        timer_out_bit_data <- ram_out_bit_data
end

shared ReadRAM is
begin
       if (ram_is_bit_addr = 0b0) then
         if (#ram_addr[7] = 0b1) then
                case ram_addr of

                (R_SP as byte) then
                        ram_in_data := SP
                |(R_DPL as byte) then
                        ram_in_data := DPL
                |(R_DPH as byte) then
                        ram_in_data := DPH
                |(R_PSW as byte) then
                        ram_in_data := PSW
                |(R_ACC as byte) then
                        ram_in_data := ACC
                |(R_B as byte) then
                        ram_in_data := B

                |(R_P0 as byte) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P0:=P0_out
                        else P0_i -> P0
                        end;
                        ram_in_data := P0
                        end
                |(R_P1 as byte) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P1:=P1_out
```

```
                        else P1_i -> P1
                        end;
                        ram_in_data := P1
                        end
                |(R_P2 as byte) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P2:=P2_out
                        else P2_i -> P2
                        end;
                        ram_in_data := P2
                        end
                |(R_TRIS0 as byte) then
                        ram_in_data := tris0
                |(R_TRIS1 as byte) then
                        ram_in_data := tris1
                |(R_TRIS2 as byte) then
                        ram_in_data := tris2

                else
                  begin
                        READ_TIMER() || timer_in_data -> ram_in_data
                  end
                end --end case
        else
                        iram_addr<-(ram_addr as Imm7) || iram_rd<-0b1 || iram_wr<-0b0 ||
                        iram_out_data->ram_in_data
       end

       else

            begin temp_addr := {0,0,0}@#ram_addr[3..7];
            if (#ram_addr[7] = 0b1) then
                case temp_addr of

                (R_SP as array8) then
                        ram_in_bit_data := #SP[(#ram_addr[0..2] as Imm3)]
                |(R_DPL as array8) then
                        ram_in_bit_data := #DPL[(#ram_addr[0..2] as Imm3)]
                |(R_DPH as array8) then
                        ram_in_bit_data := #DPH[(#ram_addr[0..2] as Imm3)]
                |(R_PSW as array8) then
                        ram_in_bit_data := #PSW[(#ram_addr[0..2] as Imm3)]
                |(R_ACC as array8) then
                        ram_in_bit_data := #ACC[(#ram_addr[0..2] as Imm3)]
                |(R_B as array8) then
                        ram_in_bit_data := #B[(#ram_addr[0..2] as Imm3)]

                |(R_P0 as array8) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P0:=P0_out
                        else P0_i -> P0
                        end;
                        ram_in_bit_data := #P0[(#ram_addr[0..2] as Imm3)]
                        end
                |(R_P1 as array8) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P1:=P1_out
                        else P1_i -> P1
                        end;
                        ram_in_bit_data := #P1[(#ram_addr[0..2] as Imm3)]
                        end
                |(R_P2 as array8) then
                        begin
--                      rmw -> rmw_int;
                        if (rmw_int = 0b1) then P2:=P2_out
                        else P2_i -> P2
                        end;
                        ram_in_bit_data := #P2[(#ram_addr[0..2] as Imm3)]
                        end
                |(R_TRIS0 as array8) then
```

```
                        ram_in_bit_data := #tris0[(#ram_addr[0..2] as Imm3)]
                  |(R_TRIS1 as array8) then
                        ram_in_bit_data := #tris1[(#ram_addr[0..2] as Imm3)]
                  |(R_TRIS2 as array8) then
                        ram_in_bit_data := #tris2[(#ram_addr[0..2] as Imm3)]
                  else
                    begin

--                      TO_TIMER() || cpu_in_bit_data -> ram_in_bit_data
                        READ_BIT_TIMER() || timer_in_bit_data -> ram_in_bit_data
                    end
                  end --end case
            else
                     begin
                        iram_addr<-(#ram_addr[3..6]@{0,1,0} as Imm7) || iram_rd<-0b1 ||
iram_wr<-0b0 ||
                        iram_out_data -> iram_out_data_int;
                        ram_in_bit_data := #iram_out_data_int[(#ram_addr[0..2] as Imm3)]
                    end
             end
end
      end --end if
end

        shared START_RD_RAM is
        begin
             ram_is_bit_addr := 0b0 ||
            ram_addr := ram_addr_tmp
        end


        shared START_RD_BIT_RAM is
        begin
             ram_is_bit_addr := 0b1 ||
            ram_addr := ram_addr_tmp
        end



shared WriteRAM is
begin

       if (ram_is_bit_addr = 0b0) then

          if (#ram_addr[7] = 0b1) then

                 case ram_addr of
                (R_SP as byte) then
                        SP:=ram_out_data
                |(R_DPL as byte) then
                        DPL:=ram_out_data
                |(R_DPH as byte) then
                        DPH:=ram_out_data
                |(R_PSW as byte) then
                        PSW:=ram_out_data
                |(R_ACC as byte) then
                        ACC:=ram_out_data
                |(R_B as byte) then
                        B:=ram_out_data

                |(R_P0 as byte) then
                        begin
                        P0_out:=ram_out_data; P0_o <- P0_out
                        end
                |(R_P1 as byte) then
                        begin
                        P1_out:=ram_out_data; P1_o <- P1_out
                        end
                |(R_P2 as byte) then
                        begin
                        P2_out:=ram_out_data; P2_o <- P2_out
                        end
```

```
                |(R_TRIS0 as byte) then
                       begin
                       tris0:=ram_out_data; tris0_o <- tris0
                       end
                |(R_TRIS1 as byte) then
                       begin
                       tris1:=ram_out_data; tris1_o <- tris1
                       end
                |(R_TRIS2 as byte) then
                       begin
                       tris2:=ram_out_data; tris2_o <- tris2
                       end

                else   begin
--                     TO_TIMER() || cpu_out_data <- ram_out_data
                       WR_TIMER()
                       end
                end --end case
          else
               iram_addr<-(ram_addr as Imm7) || iram_rd<-0b0 || iram_wr<-0b1 ||
iram_in_data<-ram_out_data
         end
       else

            begin
temp_addr := {0,0,0}@#ram_addr[3..7];
          if (#ram_addr[7] = 0b1) then

                case temp_addr of
--              case (ram_addr as array8) of

                (R_SP as array8) then
                      begin
                        temp_data := #SP;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          SP := (temp_data as byte)
                       end
                |(R_DPL as array8) then
                      begin
                        temp_data := #DPL;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          DPL := (temp_data as byte)
                       end
                |(R_DPH as array8) then
                      begin
                        temp_data := #DPH;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          DPH := (temp_data as byte)
                      end

                |(R_PSW as array8) then
                      begin
                        temp_data := #PSW;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          PSW := (temp_data as byte)
                      end
                |(R_ACC as array8) then
                      begin
                        temp_data := #ACC;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          ACC := (temp_data as byte)
                      end
                |(R_B as array8) then
                      begin
                        temp_data := #B;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          B := (temp_data as byte)
                      end

                |(R_P0 as array8) then
                      begin
                        temp_data := #P0_out;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
```

```
                        P0_out := (temp_data as byte);
                       P0_o <- P0_out
                      end
                |(R_P1 as array8) then
                      begin
                        temp_data := #P1_out;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          P1_out := (temp_data as byte);
                        P1_o <- P1_out
                      end
                |(R_P2 as array8) then
                      begin
                        temp_data := #P2_out;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          P2_out := (temp_data as byte);
                        P2_o <- P2_out
                      end
                |(R_TRIS0 as array8) then
                      begin
                        temp_data := #tris0;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          tris0 := (temp_data as byte)
                      end
                |(R_TRIS1 as array8) then
                      begin
                        temp_data := #tris1;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          tris1 := (temp_data as byte)
                      end
                |(R_TRIS2 as array8) then
                      begin
                        temp_data := #tris2;
                        temp_data[(#ram_addr[0..2] as Imm3)]:=ram_out_bit_data;
                          tris2 := (temp_data as byte)
                      end

                else   begin
--                      TO_TIMER() || cpu_out_bit_data <- ram_out_bit_data
                        WR_BIT_TIMER()
                        end
                end --end case
            else
                       begin
                        iram_addr<-(#ram_addr[3..6]@{0,1,0} as Imm7) || iram_rd<-0b1 ||
iram_wr<-0b0 ||
                        iram_out_data->iram_out_data_int;
                        temp_data := (iram_out_data_int as array8);
                        temp_data[(#ram_addr[0..2] as Imm3)] := ram_out_bit_data;
                        iram_addr<-(#ram_addr[3..6]@{0,1,0} as Imm7) || iram_wr<-0b0 ||
iram_rd<-0b1 ||
                        iram_in_data<-(temp_data as byte)
                      end
            end
          end --end if
end

;sp_out <- SP || dpl_out <- DPL || dph_out <- DPH || b_out <- B
end --end WriteRAM

        shared START_WR_RAM is
        begin
            ram_is_bit_addr := 0b0 ||
            ram_addr := ram_addr_tmp ||
           ram_out_data := ram_out_data_tmp
        end


        shared START_WR_BIT_RAM is
        begin
            ram_is_bit_addr := 0b1 ||
            ram_addr := ram_addr_tmp ||
           ram_out_bit_data := ram_out_bit_data_tmp
        end
```

```
shared mul_add is
begin
                tmp:=(tmp + (tmp_a as Imm16) as Imm16)
end

shared mul is
begin
        if (#m2[0]=0b1) then
                tmp:=(m1 as Imm16)
        else    tmp:=0
        end;

        if (#m2[1]=0b1) then
                begin
                tmp_a:={0} @ #m1 @ {0,0,0,0,0,0,0};
                mul_add()
                end
        end;

        if (#m2[2]=0b1) then
                begin
                tmp_a:={0,0} @ #m1 @ {0,0,0,0,0,0};
                mul_add()
                end
        end;

        if (#m2[3]=0b1) then
                begin
                tmp_a:={0,0,0} @ #m1 @ {0,0,0,0,0};
                mul_add()
                end
        end;

        if (#m2[4]=0b1) then
                begin
                tmp_a:={0,0,0,0} @ #m1 @ {0,0,0,0};
                mul_add()
                end
        end;

        if (#m2[5]=0b1) then
                begin
                tmp_a:={0,0,0,0,0} @ #m1 @ {0,0,0};
                mul_add()
                end
        end;

        if (#m2[6]=0b1) then
                begin
                tmp_a:={0,0,0,0,0,0} @ #m1 @ {0,0};
                mul_add()
                end
        end;

        if (#m2[7]=0b1) then
                begin
                tmp_a:={0,0,0,0,0,0,0} @ #m1 @ {0};
                mul_add()
                end
        end;
        if (tmp > 255) then
--              alu_des_ov:=0b1 || alu_des_cy:=0b0
                --reg_ov:=0b1 || reg_cy:=0b0
                PSW[2]:=0b1 || PSW[7]:=0b0
        else
--              alu_des_cy:=0b0
                --reg_cy:=0b0
                PSW[7]:=0b0
        end

end
```

```
shared div_sub is
begin
               tmp_n:=(tmp_n - (tmp_d as Imm16) as Imm16)
end


shared div_1 is
begin
       dir_d:=0b0 || pos_d:=0b00 || pos_q:=0b001;
       div_sub() || update_q();      -- 2*d is subtracted from tmp_n
       shift_d(); -- tmp_d is the original d

       if (tmp_n  < (tmp_d as Imm16)) then
              begin
              r := (tmp_n as byte)
              end
       else
              begin
              pos_q:=0b000;
              update_q() || div_sub();
              r := (tmp_n as byte)
              end
       end
end

shared div_2 is
begin
       pos_q:=0b001;
       update_q() || r:=0
end

shared div_3 is
begin
       dir_d:=0b0 || pos_d:=0b00;
       shift_d(); -- d >> 1 (in total d<<0)
       if (tmp_n  < (tmp_d as Imm16)) then --(q=12)
              begin
              r := (tmp_n as byte)
              end
       else        --(q=13)
              begin
              pos_q:=0b000;
              update_q() || div_sub();
              r := (tmp_n as byte)
              end
       end
end

shared div_44 is       --n<= q < n+4
begin
                             if (tmp_n  < (tmp_d as Imm16)) then --(4<=q<6)
                                    begin
                                    div_3()
                                    end

                             |(tmp_n  = (tmp_d as Imm16)) then --(q=6)
                                    begin
                                    div_2()
                                    end
                             else                          --(6<=q<8)
                                    begin
                                    div_1()
                                    end
                             end
end


shared div_16 is
begin
               dir_d:=0b0 || pos_d:=0b01;shift_d(); -- d >> 2 ( in total << 2 )
               if (tmp_n < (tmp_d as Imm16)) then  -- (0<= q <4)
```

```
                        begin
                        dir_d:=0b0 || pos_d:=0b00;shift_d();  -- d >> 1 ( in total << 1
)
                        div_44()
                        end

                |(tmp_n = (tmp_d as Imm16)) then --( in total d << 2 ,q=4)
                        begin
                                pos_q:=0b010;
                                update_q() || r:=0
                        end
                else                            --( in total d << 2  4 <= q < 16)
                        begin
                        dir_d:=0b1 || pos_d:=0b00;shift_d(); -- d << 1 ( in total d << 3
)
                        if (tmp_n < (tmp_d as Imm16)) then --( 4<= q <8)
                                begin
                                dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in
total d << 2)
                                div_sub();     --4*d more is subtracted from tmp_n
                                pos_q:=0b010;
                                update_q();   -- (q is 4)
                                dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in
total d<<1)

                                div_44()
                                end --end  ( 4<= q <8)

                        |(tmp_n  = (tmp_d as Imm16)) then --(q=8)

                                begin
                                pos_q:=0b011;
                                update_q() || r:=0
                                end
                        else            --(8<= q <16)
                                begin
                                pos_q:=0b011;
                                update_q(); --(q is 8)
                                div_sub(); --8*d more is subtracted from tmp_n
                                dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in
total d<<2)

                                if (tmp_n  < (tmp_d as Imm16)) then --(8<=q<12)

                                        begin
                                        dir_d:=0b0 || pos_d:=0b00;shift_d(); -- d >> 1
(in total d<<1)
                                        div_44()
                                        end

                                |(tmp_n  = (tmp_d as Imm16)) then --(q=12)
                                        begin
                                        pos_q:=0b010;
                                        update_q() || r:=0
                                        end

                                else                            --(12<=q<16)
                                        begin
                                        pos_q:=0b010;
                                        update_q(); --(q is 12)
                                        div_sub(); --4*d more is subtracted from tmp_n
                                        dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1
(in total d<<1)
                                        div_44()
                                        end
                                end
                                end
                        end
                        end
                end
end

shared div_32 is
```

```
begin
                        if (tmp_n < (tmp_d as Imm16)) then    --(0<=q<32)
                                begin
                                div_16()
                                end
                        |(tmp_n = (tmp_d as Imm16)) then      --(inc q=16)
                                begin
                                pos_q:=0b100;
                                update_q() || r:=0
                                end
                        else                                  --(16<=q<32)
                                begin
                                div_sub();     --16*d more is subtracted from tmp_n
                                pos_q:=0b100;
                                update_q();   -- (inc q=16)
                                div_16()
                                end
                        end
end

shared div_u is
begin
        tmp_n:=(n as Imm16) ||
        q:=(0 as array8) ||
        tmp_d:=(#d as array16) || dir_d:=0b1 || pos_d:=0b11;
        shift_d();  -- d << 4

        if (tmp_n < (tmp_d as Imm16)) then    --(q<16)
                begin
                div_16()
                end

        |(tmp_n = (tmp_d as Imm16)) then      --(q=16)
                begin
                pos_q:=0b100;
                update_q() || r:=0
                end
        else            --(16<=q)
                begin
                dir_d:=0b1 || pos_d:=0b01;shift_d(); -- d << 2 ( in total d << 6 )
                if (tmp_n < (tmp_d as Imm16)) then --( 16<= q <64)
                        begin
                        dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in total d <<
5)
                        if (tmp_n < (tmp_d as Imm16)) then --( 16<= q <32)
                                begin
                                dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in
total d << 4)
                                div_sub();     --16*d more is subtracted from tmp_n
                                pos_q:=0b100;
                                update_q();   -- (q is 16)
                                div_16()
                                end
                        |(tmp_n = (tmp_d as Imm16)) then --( q = 32)
                                begin
                                pos_q:=0b101;
                                update_q() || r:=0
                                end
                        else                  --(32<=q<64)
                                begin
                                div_sub();     --32*d more is subtracted from tmp_n
                                pos_q:=0b101;
                                update_q();   -- (q is 32)

                                dir_d:=0b0 || pos_d:=0b00; shift_d(); -- d >> 1 (in
total d << 4)
                                div_32()               --(32<=q<64 -> 0<=q<32)
                                end
                        end
                        end
                |(tmp_n = (tmp_d as Imm16)) then --( q = 64)
                        begin
                        pos_q:=0b110;
```

```
                        update_q() || r:=0
                        end
                else                            --(64<=q<96 (not 128, in fact
max=255/3=85)
                        begin
                        div_sub();     --64*d more is subtracted from tmp_n
                        pos_q:=0b110;
                        update_q();   -- (q is 64)

                        dir_d:=0b0 || pos_d:=0b01; shift_d(); -- d >> 2 (in total d <<
4)
                        div_32()               -- (64<=q<96 -> 0<=q<32)
                        end
                end
                end
        end
end



shared div is
begin
--      alu_des_ov:=0b0 || alu_des_cy:=0b0;
        --reg_ov:=0b0 || reg_cy:=0b0;
        PSW[2]:=0b0 || PSW[7]:=0b0;
        if      (d=0)  then
                        PSW[2]:=0b1 || q:=(CD_8 as array8) || r:=(CD_8 as byte)
                |(d=1) then
                        q:=(n as array8) || r:=0
                |(n=0) then
                        q:=(0 as array8) || r:=0
                |(n<d) then
                        q:=(0 as array8) || r:=n
                |(n=d) then
                        q:=(1 as array8) || r:=0
        else
                        case d of
                             2  then q:=(#n[1..7] as array8) || r:=(#n[0] as byte)
                            |4  then q:=(#n[2..7] as array8) || r:=(#n[0..1] as byte)
                            |8  then q:=(#n[3..7] as array8) || r:=(#n[0..2] as byte)

                            |16 then q:=(#n[4..7] as array8) || r:=(#n[0..3] as byte)
                            |32 then q:=(#n[5..7] as array8) || r:=(#n[0..4] as byte)
                            |64 then q:=(#n[6..7] as array8) || r:=(#n[0..5] as byte)
                            |128 then q:=(1 as array8) || r:=(#n[0..6] as byte)
                            else       div_u()
                        end
        end
end


shared ALU is
begin

           case alu_op_code of
             (ALU_OPC_ADD as Imm4) then begin

               v4 := ((#alu_src_1[0..3]@{0} as Imm5) +(#alu_src_2[0..3]@{0} as
Imm5)+(#alu_src_cy@{0,0,0,0} as Imm5) as Imm5);
               v8 := ((#alu_src_1[4..6]@{0} as Imm4) + (#alu_src_2[4..6]@{0} as Imm4)
+ (#v4[4..4]@{0,0,0} as Imm4) as Imm4);
               vC := ((#alu_src_1[7..7]@{0} as Imm2) + (#alu_src_2[7..7]@{0} as Imm2)
+ (#v8[3..3]@{0} as Imm2) as Imm2);

               alu_des_1[7] := #vC[0]||
               alu_des_1[6] := #v8[2]||
               alu_des_1[5] := #v8[1]||
               alu_des_1[4] := #v8[0]||
               alu_des_1[3] := #v4[3]||
               alu_des_1[2] := #v4[2]||
               alu_des_1[1] := #v4[1]||
               alu_des_1[0] := #v4[0]||
```

```
                alu_des_cy := #vC[1]||
                alu_des_ac := #v4[4]||
                alu_des_ov := #vC[1] xor #v8[3]

              end
            | (ALU_OPC_SUB as Imm4) then begin

                v4 := ((#alu_src_1[0..3]@{1} as Imm5) -(#alu_src_2[0..3]@{0} as Imm5)-
(#alu_src_cy@{0,0,0,0} as Imm5) as Imm5);
               v4_4 := (not #v4[4] as bit);
                v8 := ((#alu_src_1[4..6]@{1} as Imm4)-(#alu_src_2[4..6]@{0} as Imm4) -
(#v4_4@{0,0,0} as Imm4) as Imm4);
               v8_3 := (not #v8[3] as bit);
                vC := ((#alu_src_1[7..7]@{1} as Imm2) - (#alu_src_2[7..7]@{0} as Imm2)
- (#v8_3@{0} as Imm2) as Imm2);

                alu_des_1[7] := #vC[0]||
                alu_des_1[6] := #v8[2]||
                alu_des_1[5] := #v8[1]||
                alu_des_1[4] := #v8[0]||
                alu_des_1[3] := #v4[3]||
                alu_des_1[2] := #v4[2]||
                alu_des_1[1] := #v4[1]||
                alu_des_1[0] := #v4[0]||

                alu_des_cy := (not #vC[1] as bit)||
                alu_des_ac := (not #v4[4] as bit)||
                alu_des_ov := (not #vC[1] as bit) xor (not #v8[3] as bit)
              end

           |  (ALU_OPC_DA as Imm4) then begin

                     v := (#alu_src_1 @ {0} as Imm9);
                     if ((alu_src_ac = 0b1) or ((#v[0..3] as Imm4) > 0b1001)) then

                         v := (v + 0b000000110 as Imm9)
                    end;

                     v := (#v[0..7] @ {#v[8] or alu_src_cy} as Imm9);
                     --#v[8] := (#v[8] as bit) or alu_src_cy;

                     if((#v[8] = 0b1) or ((#v[4..7] as Imm4) > 0b1001)) then

                         v := (v + 0b001100000 as Imm9)
                    end;


                     alu_des_1 := #v[0..7] ||
                     alu_des_cy := #v[8]

              end

            |  (ALU_OPC_NOT as Imm4) then begin

                     alu_des_1[7] := (not #alu_src_1[7] as bit) ||
                     alu_des_1[6] := (not #alu_src_1[6] as bit) ||
                     alu_des_1[5] := (not #alu_src_1[5] as bit) ||
                     alu_des_1[4] := (not #alu_src_1[4] as bit) ||
                     alu_des_1[3] := (not #alu_src_1[3] as bit) ||
                     alu_des_1[2] := (not #alu_src_1[2] as bit) ||
                     alu_des_1[1] := (not #alu_src_1[1] as bit) ||
                     alu_des_1[0] := (not #alu_src_1[0] as bit)

              end
            |  (ALU_OPC_AND as Imm4) then begin

                     alu_des_1[7] := #alu_src_1[7] and #alu_src_2[7] ||
                     alu_des_1[6] := #alu_src_1[6] and #alu_src_2[6] ||
                     alu_des_1[5] := #alu_src_1[5] and #alu_src_2[5] ||
                     alu_des_1[4] := #alu_src_1[4] and #alu_src_2[4] ||
                     alu_des_1[3] := #alu_src_1[3] and #alu_src_2[3] ||
                     alu_des_1[2] := #alu_src_1[2] and #alu_src_2[2] ||
                     alu_des_1[1] := #alu_src_1[1] and #alu_src_2[1] ||
```

```
                alu_des_1[0] := #alu_src_1[0] and #alu_src_2[0]

            end
        |   (ALU_OPC_XOR as Imm4) then begin

                alu_des_1[7] := #alu_src_1[7] xor #alu_src_2[7] ||
                alu_des_1[6] := #alu_src_1[6] xor #alu_src_2[6] ||
                alu_des_1[5] := #alu_src_1[5] xor #alu_src_2[5] ||
                alu_des_1[4] := #alu_src_1[4] xor #alu_src_2[4] ||
                alu_des_1[3] := #alu_src_1[3] xor #alu_src_2[3] ||
                alu_des_1[2] := #alu_src_1[2] xor #alu_src_2[2] ||
                alu_des_1[1] := #alu_src_1[1] xor #alu_src_2[1] ||
                alu_des_1[0] := #alu_src_1[0] xor #alu_src_2[0]

            end
        |   (ALU_OPC_OR as Imm4) then begin

                alu_des_1[7] := #alu_src_1[7] or #alu_src_2[7] ||
                alu_des_1[6] := #alu_src_1[6] or #alu_src_2[6] ||
                alu_des_1[5] := #alu_src_1[5] or #alu_src_2[5] ||
                alu_des_1[4] := #alu_src_1[4] or #alu_src_2[4] ||
                alu_des_1[3] := #alu_src_1[3] or #alu_src_2[3] ||
                alu_des_1[2] := #alu_src_1[2] or #alu_src_2[2] ||
                alu_des_1[1] := #alu_src_1[1] or #alu_src_2[1] ||
                alu_des_1[0] := #alu_src_1[0] or #alu_src_2[0]

            end
        |   (ALU_OPC_RL as Imm4) then begin

               alu_des_1 := (#alu_src_1[7..7] @ #alu_src_1[0..6] as array8)

            end
        |   (ALU_OPC_RLC as Imm4) then begin

               alu_des_1 := (#alu_src_cy @ #alu_src_1[0..6] as  array8) ||
                alu_des_cy := #alu_src_1[7]

            end
        |   (ALU_OPC_RR as Imm4) then begin

               alu_des_1 := (#alu_src_1[1..7] @ {#alu_src_1[0]} as array8)

            end
        |   (ALU_OPC_RRC as Imm4) then begin

               alu_des_1 := (#alu_src_1[1..7] @ #alu_src_cy as array8) ||
                alu_des_cy := #alu_src_1[0]

            end
        |   (ALU_OPC_PCSADD as Imm4) then begin

                if (#alu_src_3[7] = 0b1 ) then
                         v16 :=((#alu_src_1 @ #alu_src_2 as Imm16) + (#alu_src_3 @
{1,1,1,1,1,1,1,1} as Imm16) as Imm16)
               else
                   v16 :=((#alu_src_1 @ #alu_src_2 as Imm16) + (#alu_src_3 @
{0,0,0,0,0,0,0,0} as Imm16) as Imm16)
               end;

                alu_des_1 := #v16[0..7] ||
                alu_des_2 := (#v16[8..15] as Imm8)

            end
        |   (ALU_OPC_PCUADD as Imm4) then begin

                v16 := ((#alu_src_1 @ #alu_src_2 as Imm16) + alu_src_3 as Imm16);
                alu_des_1 := #v16[0..7] ||
                alu_des_2 := (#v16[8..15] as Imm8)
            end
        else        begin
                alu_des_1 := (C0_8 as array8) ||
                alu_des_2 := C0_8 ||
                alu_des_cy := 0b0 ||
```

```
                    alu_des_ac := 0b0 ||
                    alu_des_ov := 0b0
                end
        end
end --end alu


-------------------------------------------------------------------------------------
----------------------------
(--
shared DEC is

begin
        rmw_int_tmp := 0b0;

        if (#op_in_int[0..4] = (ACALL as array5))    then op_out_int :=(#(OPC_ACALL as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_4 as array8))    then op_out_int :=(#(OPC_MOV_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (ADD_1 as array5))    then op_out_int :=(#(OPC_ADD_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ADD_2 as array8))    then op_out_int :=(#(OPC_ADD_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (ADD_3 as array7))    then op_out_int :=(#(OPC_ADD_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ADD_4 as array8))    then op_out_int :=(#(OPC_ADD_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (ADDC_1 as array5))   then op_out_int :=(#(OPC_ADDC_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ADDC_2 as array8))   then op_out_int :=(#(OPC_ADDC_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (ADDC_3 as array7))   then op_out_int :=(#(OPC_ADDC_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ADDC_4 as array8))   then op_out_int :=(#(OPC_ADDC_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..4] = (AJMP as array5))     then op_out_int :=(#(OPC_AJMP as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (ANL_1 as array5))    then op_out_int :=(#(OPC_ANL_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ANL_2 as array8))    then op_out_int :=(#(OPC_ANL_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (ANL_3 as array7))    then op_out_int :=(#(OPC_ANL_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ANL_4 as array8))    then op_out_int :=(#(OPC_ANL_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (ANL_5 as array8))    then op_out_int :=(#(OPC_ANL_5 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (ANL_6 as array8))    then op_out_int :=(#(OPC_ANL_6 as
Imm7) as array7) || fetch_op:=0b11 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (ANL_7 as array8))    then op_out_int :=(#(OPC_ANL_7 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (ANL_8 as array8))    then op_out_int :=(#(OPC_ANL_8 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (CJNE_1 as array8))   then op_out_int :=(#(OPC_CJNE_1 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (CJNE_2 as array8))   then op_out_int :=(#(OPC_CJNE_2 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[3..7] = (CJNE_3 as array5))   then op_out_int :=(#(OPC_CJNE_3 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[1..7] = (CJNE_4 as array7))   then op_out_int :=(#(OPC_CJNE_4 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (CLR_1 as array8))    then op_out_int :=(#(OPC_CLR_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (CLR_2 as array8))    then op_out_int :=(#(OPC_CLR_2 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (CLR_3 as array8))    then op_out_int :=(#(OPC_CLR_3 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (CPL_1 as array8))    then op_out_int :=(#(OPC_CPL_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (CPL_2 as array8))    then op_out_int :=(#(OPC_CPL_2 as
Imm7) as array7) || fetch_op:=0b00
```

```
        |  (#op_in_int[0..7] = (CPL_3 as array8))    then op_out_int :=(#(OPC_CPL_3 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (DA as array8))              then op_out_int
:=(#(OPC_DA as Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (DEC_1 as array8))    then op_out_int :=(#(OPC_DEC_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (DEC_2 as array5))    then op_out_int :=(#(OPC_DEC_2 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (DEC_3 as array8))    then op_out_int :=(#(OPC_DEC_3 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[1..7] = (DEC_4 as array7))    then op_out_int :=(#(OPC_DEC_4 as
Imm7) as array7) || fetch_op:=0b00 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (DIV as array8))      then op_out_int :=(#(OPC_DIV as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (DJNZ_1 as array5))   then op_out_int :=(#(OPC_DJNZ_1 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (DJNZ_2 as array8))   then op_out_int :=(#(OPC_DJNZ_2 as
Imm7) as array7) || fetch_op:=0b11 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (INC_1 as array8))    then op_out_int :=(#(OPC_INC_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (INC_2 as array5))    then op_out_int :=(#(OPC_INC_2 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (INC_3 as array8))    then op_out_int :=(#(OPC_INC_3 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[1..7] = (INC_4 as array7))    then op_out_int :=(#(OPC_INC_4 as
Imm7) as array7) || fetch_op:=0b00 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (INC_5 as array8))    then op_out_int :=(#(OPC_INC_5 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (JB as array8))              then op_out_int
:=(#(OPC_JB as Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (JBC as array8))      then op_out_int :=(#(OPC_JBC as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (JC as array8))              then op_out_int
:=(#(OPC_JC as Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (JMP as array8))      then op_out_int :=(#(OPC_JMP as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (JNB as array8))      then op_out_int :=(#(OPC_JNB as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (JNC as array8))      then op_out_int :=(#(OPC_JNC as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (JNZ as array8))      then op_out_int :=(#(OPC_JNZ as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (JZ as array8))              then op_out_int
:=(#(OPC_JZ as Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (LCALL as array8))    then op_out_int :=(#(OPC_LCALL as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (LJMP as array8))     then op_out_int :=(#(OPC_LJMP as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[3..7] = (MOV_1 as array5))    then op_out_int :=(#(OPC_MOV_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (MOV_2 as array8))    then op_out_int :=(#(OPC_MOV_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (MOV_3 as array7))    then op_out_int :=(#(OPC_MOV_3 as
Imm7) as array7) || fetch_op:=0b00
--      |  (#op_in_int[0..7] = (MOV_4 as array8))    then op_out_int :=(#(OPC_MOV_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (MOV_5 as array5))    then op_out_int :=(#(OPC_MOV_5 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (MOV_6 as array5))    then op_out_int :=(#(OPC_MOV_6 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (MOV_7 as array5))    then op_out_int :=(#(OPC_MOV_7 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_8 as array8))    then op_out_int :=(#(OPC_MOV_8 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (MOV_9 as array5))    then op_out_int :=(#(OPC_MOV_9 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_10 as array8))   then op_out_int :=(#(OPC_MOV_10 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[1..7] = (MOV_11 as array7))   then op_out_int :=(#(OPC_MOV_11 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_12 as array8))   then op_out_int :=(#(OPC_MOV_12 as
Imm7) as array7) || fetch_op:=0b11
```

```
        |  (#op_in_int[1..7] = (MOV_13 as array7))   then op_out_int :=(#(OPC_MOV_13 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[1..7] = (MOV_14 as array7))   then op_out_int :=(#(OPC_MOV_14 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (MOV_15 as array7))   then op_out_int :=(#(OPC_MOV_15 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_16 as array8))   then op_out_int :=(#(OPC_MOV_16 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_17 as array8))   then op_out_int :=(#(OPC_MOV_17 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (MOV_18 as array8))   then op_out_int :=(#(OPC_MOV_18 as
Imm7) as array7) || fetch_op:=0b11
        |  (#op_in_int[0..7] = (MOVC_1 as array8))   then op_out_int :=(#(OPC_MOVC_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (MOVC_2 as array8))   then op_out_int :=(#(OPC_MOVC_2 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[1..7] = (MOVX_1 as array7))   then op_out_int :=(#(OPC_MOVX_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (MOVX_2 as array8))   then op_out_int :=(#(OPC_MOVX_2 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[1..7] = (MOVX_3 as array7))   then op_out_int :=(#(OPC_MOVX_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (MOVX_4 as array8))   then op_out_int :=(#(OPC_MOVX_4 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (MUL as array8))      then op_out_int :=(#(OPC_MUL as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (NOP as array8))      then op_out_int :=(#(OPC_NOP as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (ORL_1 as array5))    then op_out_int :=(#(OPC_ORL_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ORL_2 as array8))    then op_out_int :=(#(OPC_ORL_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (ORL_3 as array7))    then op_out_int :=(#(OPC_ORL_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (ORL_4 as array8))    then op_out_int :=(#(OPC_ORL_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (ORL_5 as array8))    then op_out_int :=(#(OPC_ORL_5 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (ORL_6 as array8))    then op_out_int :=(#(OPC_ORL_6 as
Imm7) as array7) || fetch_op:=0b11 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (ORL_7 as array8))    then op_out_int :=(#(OPC_ORL_7 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (ORL_8 as array8))    then op_out_int :=(#(OPC_ORL_8 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (POP as array8))      then op_out_int :=(#(OPC_POP as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (PUSH as array8))     then op_out_int :=(#(OPC_PUSH as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (RET as array8))      then op_out_int :=(#(OPC_RET as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (RETI as array8))     then op_out_int :=(#(OPC_RETI as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (RL as array8))               then op_out_int
:=(#(OPC_RL as Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (RLC as array8))      then op_out_int :=(#(OPC_RLC as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (RR as array8))               then op_out_int
:=(#(OPC_RR as Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (RRC as array8))      then op_out_int :=(#(OPC_RRC as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (SETB_1 as array8))   then op_out_int :=(#(OPC_SETB_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (SETB_2 as array8))   then op_out_int :=(#(OPC_SETB_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (SJMP as array8))     then op_out_int :=(#(OPC_SJMP as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[3..7] = (SUBB_1 as array5))   then op_out_int :=(#(OPC_SUBB_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (SUBB_2 as array8))   then op_out_int :=(#(OPC_SUBB_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (SUBB_3 as array7))   then op_out_int :=(#(OPC_SUBB_3 as
Imm7) as array7) || fetch_op:=0b00
```

```
        |  (#op_in_int[0..7] = (SUBB_4 as array8))   then op_out_int :=(#(OPC_SUBB_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (SWAP as array8))     then op_out_int :=(#(OPC_SWAP as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (XCH_1 as array5))    then op_out_int :=(#(OPC_XCH_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (XCH_2 as array8))    then op_out_int :=(#(OPC_XCH_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (XCH_3 as array7))    then op_out_int :=(#(OPC_XCH_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[1..7] = (XCHD as array7))     then op_out_int :=(#(OPC_XCHD as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[3..7] = (XRL_1 as array5))    then op_out_int :=(#(OPC_XRL_1 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (XRL_2 as array8))    then op_out_int :=(#(OPC_XRL_2 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[1..7] = (XRL_3 as array7))    then op_out_int :=(#(OPC_XRL_3 as
Imm7) as array7) || fetch_op:=0b00
        |  (#op_in_int[0..7] = (XRL_4 as array8))    then op_out_int :=(#(OPC_XRL_4 as
Imm7) as array7) || fetch_op:=0b10
        |  (#op_in_int[0..7] = (XRL_5 as array8))    then op_out_int :=(#(OPC_XRL_5 as
Imm7) as array7) || fetch_op:=0b10 || rmw_int_tmp := 0b1
        |  (#op_in_int[0..7] = (XRL_6 as array8))    then op_out_int :=(#(OPC_XRL_6 as
Imm7) as array7) || fetch_op:=0b11 || rmw_int_tmp := 0b1
        else op_out_int := (#(OPC_ERROR as Imm7) as array7) || fetch_op:=0b00
         end --end if
end --end X8051_DEC
--)



        shared SET_PSW is
        begin

reg_cy:=#reg_psw[7]||reg_ac:=#reg_psw[6]||reg_f0:=#reg_psw[5]||reg_rs1:=#reg_psw[4]||

reg_rs0:=#reg_psw[3]||reg_ov:=#reg_psw[2]||reg_nu:=#reg_psw[1]||reg_p:=#reg_psw[0]
        end


        shared START_RD_XRAM is
        begin
             xram_rd <- 0b1||
            tris_data <-0b0||
            xram_e <- 0b1 ||
            xram_addr <- xram_addr_tmp
        end

         shared START_WR_XRAM is
         begin
             xram_wr <- 0b1||
            tris_data <-0b1||
            xram_e <- 0b1 ||
            xram_addr <- xram_addr_tmp ||
             xram_out_data <- xram_out_data_tmp
         end


        shared InstrFetch is
        begin
           rom_addr <- reg_pc || rom_rd <-0b1 ||  rom_data -> reg_ir
        end

        shared PcInc is
        begin
           reg_pc_plus:=(reg_pc+1 as Imm16)
        end

        shared SET_PC is
        begin
            reg_pc:=(#reg_pcl @ #reg_pc_8_10 @ #reg_pc_11_15 as Imm16)
        end
```

```
        shared X8051_Fetch is
        begin
                        exe_opcode := (#op_out_int[0..6] as Imm7) ||
                        reg_acc:=ACC || reg_psw := PSW;
           case (fetch_op) of
                0b10    then    begin
                                        InstrFetch() || PcInc();
                                        reg_op2:=reg_ir || reg_pc:=reg_pc_plus ||
SET_PSW()
                                        end
               |0b11    then
                                begin
                                        InstrFetch() || PcInc();
                                        reg_op2:=reg_ir || reg_pc:=reg_pc_plus;
                                        InstrFetch() || PcInc();
                                        reg_op3:=reg_ir || reg_pc:=reg_pc_plus ||
SET_PSW()
                                end
                else    SET_PSW()
          end
        end


-------------------------------------------------------------------------------------
------------------------------


shared Interrupt is
begin

int_req->int_req_int || int_mask->int_mask_int
--below is for testing only
|| int_lproc<-int_lproc_int
---------------------------
;

                        if (int_req_int=0b000 or ((int_mask_int=0b11) and
(int_hproc_int=0b1))
or ((int_mask_int=0b10) and (int_hproc_int=0b1)) or ((int_mask_int=0b01) and
(int_lproc_int=0b1)))
                                then    int_lproc_int:=0b0 || int_hproc_int:=0b0 --not
required actually
                                        -- cpu_state:=0b01
                        else
                        begin
                        if (int_mask_int=0b01) then int_lproc_int:=0b1
                        else int_hproc_int:=0b1
                        end;

                        data_bus := SP;
                        alu_src_1:= data_bus || alu_op_code:=(ALU_OPC_ADD as Imm4) ||
alu_src_2:=(C0_8 as byte) ||
                        alu_src_cy:=0b1; ALU();
                        ram_out_data_tmp := (#reg_pc[0..7] as byte) || ram_addr_tmp :=
(alu_des_1 as byte);
                        START_WR_RAM(); WriteRAM();

                        alu_src_1:=(alu_des_1 as byte) || alu_op_code:=(ALU_OPC_ADD as
Imm4) || alu_src_2:=(C0_8 as byte) || alu_src_cy:=0b1; ALU();
                        ram_out_data_tmp := (#reg_pc[8..15] as byte) || ram_addr_tmp :=
(alu_des_1 as byte);
                        START_WR_RAM(); WriteRAM();
                        SP := (alu_des_1 as byte);


                        case (int_req_int) of

                          0b001 then
                                begin
(-- change to || not working
```

```
reg_pc:=(#ISR_1 @ #C0_8 as Imm16) ||
ram_addr_tmp :=0b10001001 || ram_out_bit_data_tmp :=0b0;
--)
                                        reg_pc:=(#ISR_1 @ #C0_8 as Imm16);
                                        ram_addr_tmp :=0b10001001 || ram_out_bit_data_tmp
:=0b0;
                                        START_WR_BIT_RAM(); WriteRAM()
                                end

                        | 0b010 then
                                begin
                                        reg_pc:=(#ISR_2 @ #C0_8 as Imm16);
                                        ram_addr_tmp :=0b10001101 || ram_out_bit_data_tmp
:=0b0;
                                        START_WR_BIT_RAM(); WriteRAM()
                                end

                        | 0b011 then
                                begin
                                        reg_pc:=(#ISR_3 @ #C0_8 as Imm16);
                                        ram_addr_tmp :=0b10001011 || ram_out_bit_data_tmp
:=0b0;
                                        START_WR_BIT_RAM(); WriteRAM()
                                end
                        | 0b100 then
                                begin
                                        reg_pc:=(#ISR_4 @ #C0_8 as Imm16);
                                        ram_addr_tmp :=0b10001111 || ram_out_bit_data_tmp
:=0b0;
                                        START_WR_BIT_RAM(); WriteRAM()
                                end

                          else
                                begin
                                        reg_pc:=(#ISR_5 @ #C0_8 as Imm16)
                                end
                        end --end case

                        -- cpu_state := 0b01
                        end
                        end --end if

end -- end of interrupt



-------------------------------------------------------------------------------
shared EXE is
begin


                    case (exe_opcode) of

-------------------------------------------------------------------------------

                          --ACALL addr11
                          -- sp       <- sp + 1
                          -- mem(sp)  <- pc(7-0)
                          -- sp       <- sp + 1
                          -- mem(sp)  <- pc(15-8)
                          -- pc(10-0) <- page address
                          --
                          (OPC_ACALL as Imm7)  then

                        begin

                        data_bus := SP;
                        alu_src_1 := data_bus || alu_op_code:=(ALU_OPC_ADD as Imm4) ||
alu_src_2:=(C0_8 as byte) ||
                        alu_src_cy:=0b1; ALU();
```

```
                      ram_out_data_tmp := (#reg_pc[0..7] as byte) || ram_addr_tmp :=
(alu_des_1 as byte);
                      START_WR_RAM() ||
                      alu_src_1 := (alu_des_1 as byte) || alu_op_code:=(ALU_OPC_ADD as
Imm4) || alu_src_2:=(C0_8 as byte) ||
                      alu_src_cy:=0b1; ALU() || WriteRAM();

                      ram_out_data_tmp := (#reg_pc[8..15] as byte) || ram_addr_tmp :=
(alu_des_1 as byte);

                      START_WR_RAM(); WriteRAM();

                      SP := (alu_des_1 as byte);

                      reg_pc_8_10:=(#reg_op1[5..7] as Imm3) ||
reg_pc_11_15:=(#reg_pc[11..15] as Imm5) || reg_pcl:=reg_op2;

                      SET_PC()
-- cpu_state:=0b00
                        end

-------------------------------------------------------------------------------
                        --ADD A,Rn
                        -- acc <- acc + (r)
                        --
                      | (OPC_ADD_1 as Imm7)  then

                      begin
                       GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc||
alu_src_2:=ram_in_data|| alu_src_cy:=0b0; ALU();

                        ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                               GET_PSW()  -- cpu_state:=0b00
                        end

                        --ADD A,direct
                        -- acc <- acc + (direct)
                        --
                      | (OPC_ADD_2 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=ram_in_data ||
                      alu_src_cy := 0b0; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                      GET_PSW() -- cpu_state:=0b00
                      end

                        --ADD A,((r))
                        -- acc <- acc + @Ri
                        --
                      | (OPC_ADD_3 as Imm7) then

                      begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;
                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=ram_in_data ||
                      alu_src_cy := 0b0; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
```

```
                      GET_PSW() -- cpu_state:=0b00
                      end

                        --ADD A,#data
                        -- acc <- acc + #data
                        --
                      | (OPC_ADD_4 as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=reg_op2 ||
                      alu_src_cy := 0b0; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                      GET_PSW() -- cpu_state:=0b00
                      end

                        --ADDC A,Rn
                        -- acc <- acc + cy + (r)
                        --
                      | (OPC_ADDC_1 as Imm7) then

                      begin
                      GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc||
alu_src_2:=ram_in_data||
                      alu_src_cy:=reg_cy; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                      GET_PSW() -- cpu_state:=0b00
                        end

                        --ADDC A,direct
                        -- acc <- acc + cy + (direct)
                        --
                      | (OPC_ADDC_2 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=ram_in_data ||
                      alu_src_cy := reg_cy; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                      GET_PSW() -- cpu_state:=0b00
                      end

                        --ADDC A,((r))
                        -- acc <- acc + cy + @Ri
                        --
                      | (OPC_ADDC_3 as Imm7) then

                      begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;
                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=ram_in_data ||
                      alu_src_cy := reg_cy; ALU();

                      ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                      GET_PSW() -- cpu_state:=0b00
                      end
```

```
                        --ADDC A,#data
                        -- acc <- acc + cy + #data
                        --
                       | (OPC_ADDC_4 as Imm7) then

                       begin
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=reg_op2 ||
                       alu_src_cy := reg_cy; ALU();

                       ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy ||
reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                       GET_PSW() -- cpu_state:=0b00
                       end

                        --AJMP addr11
                        -- pc(10-0) <- page address
                        --
                       | (OPC_AJMP as Imm7) then

                       begin
                       reg_pc_8_10:=(#reg_op1[5..7] as Imm3) ||
reg_pc_11_15:=(#reg_pc[11..15] as Imm5) || reg_pcl:=reg_op2;
                        SET_PC()
                       -- cpu_state:=0b00
                       end

                        --ANL A,Rn   logical &&
                        -- acc <- acc && (r)
                        --
                       | (OPC_ANL_1 as Imm7) then

                       begin
                       GET_RAM_ADDR_1();
                       START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_acc||
alu_src_2:=ram_in_data; ALU();

                        ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                        end

                        --
                        -- acc <- acc && (direct)
                        --
                       | (OPC_ANL_2 as Imm7) then

                       begin
                       ram_addr_tmp := reg_op2;
                       START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_acc||
alu_src_2:=ram_in_data; ALU();

                        ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                        end

                        --
                        -- acc <- acc && ((r))
                        --
                       | (OPC_ANL_3 as Imm7) then

                       begin
                       GET_RAM_ADDR_2();
                       START_RD_RAM(); ReadRAM();
                       data_bus:=ram_in_data;

                       ram_addr_tmp := data_bus;
                       START_RD_RAM(); ReadRAM();
                       alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=ram_in_data; ALU();

                        ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
```

```
                        end

                          --
                          -- acc <- acc && #data
                          --
                        | (OPC_ANL_4 as Imm7) then

                        begin
                        alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=reg_op2; ALU();

                          ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                        end

                          --
                          -- (direct) <- (direct) && acc
                          --
                        | (OPC_ANL_5 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();

                          alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_acc||
alu_src_2:=ram_in_data; ALU();

                          ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
reg_op2;
                                START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                          end

                          --
                          -- (direct) <- (direct) && #data
                          --
                        | (OPC_ANL_6 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                        alu_op_code:=(ALU_OPC_AND as Imm4) || alu_src_1:=reg_op3 ||
alu_src_2:=ram_in_data; ALU();

                          ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
reg_op2;
                                START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                          --
                          -- cy <- cy & (bit)
                          --
                        | (OPC_ANL_7 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_BIT_RAM(); ReadRAM();
                        data_bit:=ram_in_bit_data;
                        reg_cy:=(reg_cy and data_bit);

                        GET_PSW() -- cpu_state:=0b00
                        end

                          --
                          -- cy <- cy & ~(bit)
                          --
                        | (OPC_ANL_8 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_BIT_RAM(); ReadRAM();
                        data_bit:=ram_in_bit_data;
                        reg_cy:=(reg_cy and (not data_bit));

                        GET_PSW() -- cpu_state:=0b00
```

```
                        end

                          --CJNE A,dir,rel
                          -- if( a != (direct) )
                          --     pc <- pc + rel
                          -- if( a < (direct) )
                          --     cy <- 1
                          -- else
                          --     cy <- 0
                          --
                        | (OPC_CJNE_1 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;

                        if (reg_acc /= data_bus) then
                          begin
                            alu_op_code:=(ALU_OPC_PCSADD as Imm4) || alu_src_1:=
(#reg_pc[0..7] as byte)
                           || alu_src_2 := (#reg_pc[8..15] as byte) || alu_src_3 :=
reg_op3; ALU() ||
                          if (reg_acc < data_bus) then reg_cy := 0b1
                            else reg_cy := 0b0 end;

                          reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                          reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                            SET_PC() ||
                        GET_PSW()
                          end
                          --else -- cpu_state:=0b00
                        end
                          end

                          --CJNE A,#data,rel
                          -- if( a != #data )
                          --     pc <- pc + rel
                          -- if( a < #data )
                          --     cy <- 1
                          -- else
                          --     cy <- 0
                          --
                        | (OPC_CJNE_2 as Imm7) then

                        begin
                        if (reg_acc /= reg_op2) then
                          begin
                            alu_op_code:=(ALU_OPC_PCSADD as Imm4) || alu_src_1:=
(#reg_pc[0..7] as byte)
                           || alu_src_2 := (#reg_pc[8..15] as byte) || alu_src_3 :=
reg_op3; ALU() ||
                          if (reg_acc < reg_op2) then reg_cy := 0b1
                            else reg_cy := 0b0 end;

                          reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                          reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);
                            SET_PC() ||
                        GET_PSW()                        end
                          --else -- cpu_state:=0b00
                        end
                          end

                          --CJNE Rn,#data,rel
                          -- if( (r) != #data )
                          --     pc <- pc + rel
                          -- if( (r) < #data )
                          --     cy <- 1
                          -- else
                          --     cy <- 0
```

```
                        --
                      | (OPC_CJNE_3 as Imm7) then

                       begin
                       GET_RAM_ADDR_1();
                      START_RD_RAM(); ReadRAM();
                    data_bus:=ram_in_data;

                      if (data_bus /= reg_op2) then
                        begin
                          alu_op_code:=(ALU_OPC_PCSADD as Imm4) || alu_src_1:=
(#reg_pc[0..7] as byte)
                         || alu_src_2 := (#reg_pc[8..15] as byte) || alu_src_3 :=
reg_op3; ALU() ||
                        if (data_bus < reg_op2) then reg_cy := 0b1
                          else reg_cy := 0b0 end;

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC() ||
                      GET_PSW()
                        end
                       --else -- cpu_state:=0b00
                      end
                       end

                       --CJNE @Ri,#data,rel
                       -- if( ((r)) != #data )
                       --     pc <- pc + rel
                       -- if( ((r)) < #data )
                       --     cy <- 1
                       -- else
                       --     cy <- 0
                       --
                      | (OPC_CJNE_4 as Imm7) then

                       begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;

                      ram_addr_tmp:=data_bus;
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;

                      if (data_bus /= reg_op2) then
                        begin
                          alu_op_code:=(ALU_OPC_PCSADD as Imm4) || alu_src_1:=
(#reg_pc[0..7] as byte)
                         || alu_src_2 := (#reg_pc[8..15] as byte) || alu_src_3 :=
reg_op3; ALU() ||
                        if (data_bus < reg_op2) then reg_cy := 0b1
                          else reg_cy := 0b0 end;

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC() ||
                      GET_PSW()
                        end
                       --else -- cpu_state:=0b00
                      end
                       end

                       --CLR A
                       -- acc <- 0
                       --
                      | (OPC_CLR_1 as Imm7) then

                       begin
```

```
                        ACC := C0_8  -- cpu_state:=0b00
                         end

                         --CLR C
                         -- cy <- 0
                         --
                        | (OPC_CLR_2 as Imm7) then

                        begin
                        reg_cy:=0b0;
                        GET_PSW() -- cpu_state:=0b00
                         end

                         --CLR bit
                         -- (bit) <- 0
                         --
                        | (OPC_CLR_3 as Imm7) then

                        begin
                        ram_out_bit_data_tmp :=0b0 || ram_addr_tmp :=reg_op2;
                        START_WR_BIT_RAM(); WriteRAM() -- cpu_state:=0b00
                         end

                         --CPL A    logically complements each bit of the Accumulator
                         -- acc <- ~acc
                         --
                        | (OPC_CPL_1 as Imm7) then

                        begin
                        alu_op_code:=(ALU_OPC_NOT as Imm4) || alu_src_1:=reg_acc;
ALU();

                        ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                        end

                         --CPL C
                         -- cy <- ~cy
                         --
                        | (OPC_CPL_2 as Imm7) then

                        begin
                        reg_cy:=(not reg_cy);
                        GET_PSW() -- cpu_state:=0b00
                        end

                         --CPL bit
                         -- (bit) <- ~(bit)
                         --
                        | (OPC_CPL_3 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_BIT_RAM(); ReadRAM();
                        data_bit:=ram_in_bit_data;

                        ram_addr_tmp :=reg_op2 || ram_out_bit_data_tmp := (not
data_bit);
                        START_WR_BIT_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                         --DA A   Decimal-adjust Accumulator for Addition
                         -- see I8051_ALU
                         --
                        | (OPC_DA as Imm7) then

                        begin
                        alu_op_code:=(ALU_OPC_DA as Imm4) || alu_src_1:=reg_acc ||
alu_src_cy:=reg_cy || alu_src_ac:=reg_ac; ALU();

                        ACC := (alu_des_1 as byte) || reg_cy:=alu_des_cy;
                        GET_PSW()  -- cpu_state:=0b00
                        end
```

```
                        --DEC A
                        -- acc <- acc - 1
                        --
                       | (OPC_DEC_1 as Imm7) then

                       begin
                       alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                       ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                       end

                        --DEC Rn
                        -- (r) <- (r) - 1
                        --
                       | (OPC_DEC_2 as Imm7) then

                       begin
                       GET_RAM_ADDR_1();
                       START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                       ram_out_data_tmp :=(alu_des_1 as byte); START_WR_RAM();
WriteRAM() -- cpu_state:=0b00
                       end

                        --DEC direct
                        -- (direct) <- (direct) - 1
                        --
                       | (OPC_DEC_3 as Imm7) then

                       begin
                       ram_addr_tmp := reg_op2;
                       START_RD_RAM(); ReadRAM();
                        alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_2:=C0_8 ||
alu_src_1:=ram_in_data || alu_src_cy:=0b1; ALU();

                        ram_out_data_tmp :=(alu_des_1 as byte) || ram_addr_tmp :=
reg_op2;
                              START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --   DEC @Ri
                        -- ((r)) <- ((r)) - 1
                        --
                       | (OPC_DEC_4 as Imm7) then

                       begin
                       GET_RAM_ADDR_2();
                       START_RD_RAM(); ReadRAM();
                       data_bus:=ram_in_data; --data_bus contains the content of Ri

                       ram_addr_tmp := data_bus;
                       START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                       ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
data_bus;
                       START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                       end

                        --DJNZ Rn,rel
                        -- (r) <- (r) - 1
                        -- if( (r) != 0 )
                        --     pc <- pc + rel
                        --
                       | (OPC_DJNZ_1 as Imm7) then

                       begin
```

```
                      GET_RAM_ADDR_1();
                      START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp :=(alu_des_1 as byte);
                      START_WR_RAM() || data_bus:=(alu_des_1 as byte); WriteRAM();

                      if (data_bus /= C0_8) then
                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte)
                        || alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();
                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()
                        end
                       --else -- cpu_state:=0b00
                      end
                      end

                        --DJNZ direct,rel
                        -- (direct) <- (direct) - 1
                        -- if( (direct) != 0 )
                        --     pc <- pc + rel
                        --
                      | (OPC_DJNZ_2 as Imm7) then

                       begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
reg_op2;
                      START_WR_RAM() || data_bus:=(alu_des_1 as byte); WriteRAM();

                      if (data_bus /= C0_8) then
                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte)
                        || alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op3;
ALU();
                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()                    end
                       --else -- cpu_state:=0b00
                      end
                      end

                        --INC A
                        -- acc <- acc + 1
                        --
                      | (OPC_INC_1 as Imm7) then

                       begin
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=reg_acc ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ACC := (alu_des_1 as byte)  -- cpu_state:=0b00
                      end

                        --INC Rn
                        -- (r) <- (r) + 1
                        --
```

```
                      | (OPC_INC_2 as Imm7) then

                       begin
                      GET_RAM_ADDR_1();
                      START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp :=(alu_des_1 as byte); START_WR_RAM();
WriteRAM() -- cpu_state:=0b00
                      end

                        --INC direct
                        -- (direct) <- (direct) + 1
                        --
                      | (OPC_INC_3 as Imm7) then

                       begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
reg_op2;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                      end

                        -- INC @Ri
                        -- ((r)) <- ((r)) + 1
                        --
                        --
                      | (OPC_INC_4 as Imm7) then

                       begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data; --data_bus contains the content of Ri

                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();

                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2:=C0_8 || alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp := (alu_des_1 as byte) || ram_addr_tmp :=
data_bus;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                      end

                        --INC DPTR
                        -- dptr <- dptr + 1
                        --
                      | (OPC_INC_5 as Imm7) then

                       begin
--                    ram_addr_tmp := R_DPL;
--                    START_RD_RAM(); ReadRAM();
                      data_bus := DPL;
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=data_bus ||
alu_src_2:=C0_8 || alu_src_cy:=0b1;
ALU();
--                    ram_addr_tmp :=R_DPL || ram_out_data_tmp :=(alu_des_1 as byte);
START_WR_RAM(); WriteRAM();
                      DPL := (alu_des_1 as byte);

--                    ram_addr_tmp := R_DPH;
--                    START_RD_RAM(); ReadRAM();
                      data_bus := DPH;
                       alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=data_bus ||
alu_src_2:=C0_8 || alu_src_cy:=alu_des_cy; ALU();
                      DPH := (alu_des_1 as byte) -- cpu_state:=0b00
```

```
                        end

                          --JB blt,rel
                          -- if( (bit) == 1 )
                          --     pc <- pc + rel
                          --
                        | (OPC_JB as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_BIT_RAM(); ReadRAM();
                        data_bit:=ram_in_bit_data;
                        if (data_bit = 0b1) then
                          begin
                          alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                          alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op3;
ALU();
                          reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                          reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                            SET_PC()                    end
                        --else -- cpu_state:=0b00
                        end
                        end

                          --JBC bit,rel
                          -- if( (bit) == 1 )
                          --     pc <- pc + rel
                          --     (bit) <- 0
                          --
                        | (OPC_JBC as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_BIT_RAM(); ReadRAM();
                        data_bit:=ram_in_bit_data;
                        if (data_bit = 0b1) then
                          begin
                          alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                          alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op3;
ALU() ||
                          ram_out_bit_data_tmp :=0b0 || ram_addr_tmp :=reg_op2;
                          START_WR_BIT_RAM(); WriteRAM();

                          reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                          reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                            SET_PC()
                          end
                        --else -- cpu_state:=0b00
                        end
                        end

                          --JC rel
                          -- if( cy == 1 )
                          --     pc <- pc + rel
                          --
                        | (OPC_JC as Imm7) then

                        begin
                        if (reg_cy = 0b1) then
                          begin
                          alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                          alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();

                          reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
```

```
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()
                        end
                      --else -- cpu_state:=0b00
                      end
                      end

                        --JMP @A+DPTR
                        -- pc <- dptr + acc
                        --
                      | (OPC_JMP as Imm7) then

                      begin
                      data_bus:=DPH;
                      alu_src_2:=data_bus;

                      data_bus:=DPL;
                      alu_op_code:=(ALU_OPC_PCSADD as Imm4) || alu_src_1:=data_bus ||
                      alu_src_3:=reg_acc; ALU();

                      reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                      reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                        SET_PC() -- cpu_state:=0b00
                      end

                        --JNB bit,rel
                        -- if( (bit) == 0 )
                        --     pc <- pc + rel
                        --
                      | (OPC_JNB as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_BIT_RAM(); ReadRAM();
                      data_bit:=ram_in_bit_data;

                      if (data_bit = 0b0) then
                        begin
                         alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                        alu_src_2:=(#reg_pc[8..15] as byte) ||  alu_src_3:=reg_op3;
ALU();

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()                    end
                      --else -- cpu_state:=0b00
                      end
                      end

                        --JNC rel
                        -- if( cy == 0 )
                        --     pc <- pc + rel
                        --
                      | (OPC_JNC as Imm7) then

                      begin
                      if (reg_cy = 0b0) then
                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                        alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);
```

```
                          SET_PC()
                        end
                       --else -- cpu_state:=0b00
                      end
                      end

                        --JNZ rel
                        -- if( acc != 0 )
                        --     pc <- pc + rel
                        --
                      | (OPC_JNZ as Imm7) then

                       begin
                      if (reg_acc /= C0_8) then
                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                        alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()
                        end
                       --else -- cpu_state:=0b00
                      end
                      end

                        --JZ rel
                        -- if( acc == 0 )
                        --     pc <- pc + rel
                        --
                      | (OPC_JZ as Imm7) then

                       begin
                      if (reg_acc = C0_8) then
                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                        alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();

                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()                    end
                       --else -- cpu_state:=0b00
                      end
                      end

                        --LCALL addr16
                        -- sp       <- sp + 1
                        -- mem(sp)  <- pc(7-0)
                        -- sp       <- sp + 1
                        -- mem(sp)  <- pc(15-8)
                        -- pc(15-0) <- address
                        --
                      | (OPC_LCALL as Imm7) then

                      begin
                        data_bus := SP;
                      alu_src_1:= data_bus ||alu_op_code:=(ALU_OPC_ADD as
Imm4)||alu_src_2:=(C0_8 as byte)||
                      alu_src_cy:=0b1; ALU();

                      ram_out_data_tmp := (#reg_pc[0..7] as byte) || ram_addr_tmp
:=(alu_des_1 as byte);
                      START_WR_RAM() ||
```

```
                      alu_src_1:=(alu_des_1 as byte) || alu_op_code:=(ALU_OPC_ADD as
Imm4) || alu_src_2:=(C0_8 as byte) ||
                      alu_src_cy:=0b1; ALU() || WriteRAM();

                      ram_out_data_tmp := (#reg_pc[8..15] as byte) || ram_addr_tmp
:=(alu_des_1 as byte);
                      START_WR_RAM(); WriteRAM();
                      SP :=(alu_des_1 as byte);
                      reg_pc_8_10:=(#reg_op2[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_op2[3..7] as Imm5) || reg_pcl:=reg_op3;
                      SET_PC()
                      -- cpu_state:=0b00
                        end

                        --LJMP addr16
                        -- pc(15-0) <- address
                        --
                      | (OPC_LJMP as Imm7) then

                      begin
                      reg_pc_8_10:=(#reg_op2[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_op2[3..7] as Imm5) || reg_pcl:=reg_op3;
                        SET_PC()
                      -- cpu_state:=0b00
                        end


                        --MOV A,Rn
                        -- acc <- (r)
                        --
                      |(OPC_MOV_1 as Imm7) then

                      begin
                      GET_RAM_ADDR_1();
                      START_RD_RAM(); ReadRAM();
                      ACC := ram_in_data
-- cpu_state:=0b00
                        end

                        --MOV A,direct
                        -- acc <- (direct)
                        --
                      | (OPC_MOV_2 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                      ACC := ram_in_data
-- cpu_state:=0b00
                        end



                        --MOV A,@Ri
                        -- acc <- ((r))
                        --
                      | (OPC_MOV_3 as Imm7) then

                      begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;
                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                      ACC := ram_in_data
-- cpu_state:=0b00
                        end


                        --MOV A,#data
                        -- acc <- #data
                        --
                      | (OPC_MOV_4 as Imm7) then
```

```
                        begin
                        ACC := reg_op2 -- cpu_state:=0b00
-- || got problem       ACC := reg_op2 || -- cpu_state:=0b00
                         end


                         --MOV Rn,A
                         -- (r) <- acc
                         --
                        | (OPC_MOV_5 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        ram_out_data_tmp := reg_acc; START_WR_RAM(); WriteRAM() --
cpu_state:=0b00
                         end

                         --MOV Rn,direct
                         -- (r) <- (direct)
                         --
                        | (OPC_MOV_6 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;

                        GET_RAM_ADDR_1();
                        ram_out_data_tmp := data_bus; START_WR_RAM(); WriteRAM() --
cpu_state:=0b00
                         end

                         --MOV Rn,#data
                         -- (r) <- #data
                         --
                        | (OPC_MOV_7 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        ram_out_data_tmp := reg_op2; START_WR_RAM(); WriteRAM() --
cpu_state:=0b00
                         end


                         --MOV direct,A
                         -- (direct) <- acc
                         --
                        | (OPC_MOV_8 as Imm7) then

                        begin
                        ram_out_data_tmp := reg_acc || ram_addr_tmp := reg_op2;
                        START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                         end

                         --MOV direct,Rn
                         -- (direct) <- (r)
                         --
                        | (OPC_MOV_9 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;

                        ram_out_data_tmp := data_bus || ram_addr_tmp := reg_op2;
                        START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                         end

                         --MOV direct,direct
                         -- (direct) <- (direct)
                         --
                        | (OPC_MOV_10 as Imm7) then
```

```
begin
ram_addr_tmp := reg_op2;
START_RD_RAM(); ReadRAM();
data_bus:=ram_in_data;

ram_out_data_tmp := data_bus || ram_addr_tmp := reg_op3;
START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
  end

  --MOV direct,@Ri
  -- (direct) <- ((r))
  --
| (OPC_MOV_11 as Imm7) then

begin
GET_RAM_ADDR_2();
START_RD_RAM(); ReadRAM();
data_bus:=ram_in_data;
ram_addr_tmp := data_bus;
START_RD_RAM(); ReadRAM();
data_bus:=ram_in_data;

ram_out_data_tmp := data_bus || ram_addr_tmp := reg_op2;
START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
  end

  --MOV direct,#data
  -- (direct) <- #data
  --
| (OPC_MOV_12 as Imm7) then

begin
ram_out_data_tmp := reg_op3 || ram_addr_tmp := reg_op2;
START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
  end


  --MOV @Ri,A
  -- ((r)) <- acc
  --
| (OPC_MOV_13 as Imm7) then

begin
GET_RAM_ADDR_2();
START_RD_RAM(); ReadRAM();
data_bus:=ram_in_data;

ram_out_data_tmp := reg_acc || ram_addr_tmp := data_bus;
START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
  end

  --MOV @Ri,direct
  -- ((r)) <- (direct)
  --
| (OPC_MOV_14 as Imm7) then

begin
GET_RAM_ADDR_2();
START_RD_RAM(); ReadRAM();
data_bus:=ram_in_data;

ram_addr_tmp := reg_op2;
START_RD_RAM(); ReadRAM();
ram_addr_tmp := data_bus;
data_bus:=ram_in_data;

ram_out_data_tmp := data_bus;
START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
  end

  --MOV @Ri,#data
  -- ((r)) <- #data
```

```
                        --
                       | (OPC_MOV_15 as Imm7) then

                       begin
                       GET_RAM_ADDR_2();
                       START_RD_RAM(); ReadRAM();
                       data_bus:=ram_in_data;

                       ram_out_data_tmp := reg_op2 || ram_addr_tmp := data_bus;
                       START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --MOV C,bit
                        -- cy <- (bit)
                        --
                       | (OPC_MOV_16 as Imm7) then

                       begin
                       ram_addr_tmp := reg_op2;
                       START_RD_BIT_RAM(); ReadRAM();
                       reg_cy:=ram_in_bit_data;

                       GET_PSW() -- cpu_state:=0b00
                        end

                        --MOV bit,C
                        -- (bit) <- cy
                        --
                       | (OPC_MOV_17 as Imm7) then

                       begin
                       ram_addr_tmp :=reg_op2 || ram_out_bit_data_tmp :=reg_cy;
                       START_WR_BIT_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --MOV DPTR,#data16
                        -- dph <- #data-hi
                        -- dpl <- #data-lo
                        --
                       | (OPC_MOV_18 as Imm7) then

                       begin
                       DPH := reg_op2 || DPL:=reg_op3
                       -- cpu_state:=0b00
                        end
(--
                        --MOVC A,@A+DPTR
                        -- acc <- (dptr + acc)
                        --
                       | (OPC_MOVC_1 as Imm7) then

                       begin
                       data_bus:=DPH; alu_src_2 := data_bus;
                       data_bus:=DPL;

                       alu_op_code:=(ALU_OPC_PCUADD as Imm4) || alu_src_1:=data_bus ||
                       alu_src_3 := reg_acc || reg_pc_tmp := reg_pc; ALU();

                       reg_pc := (#(alu_des_1 as byte) @ #alu_des_2 as Imm16);
                       InstrFetch();
                       ACC := reg_ir || reg_pc := reg_pc_tmp
                       -- cpu_state:=0b00
                        end

                        --MOVC A,@A+PC
                        -- acc <- (pc + acc)
                        --
                       | (OPC_MOVC_2 as Imm7) then

                       begin
                       alu_op_code:=(ALU_OPC_PCUADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
```

```
                      alu_src_2 := (#reg_pc[8..15] as byte) || alu_src_3 := reg_acc ||
reg_pc_tmp := reg_pc; ALU();

                      reg_pc := (#(alu_des_1 as byte) @ #alu_des_2 as Imm16);
                      InstrFetch();
                      ACC := reg_ir || reg_pc := reg_pc_tmp
                      -- cpu_state:=0b00
                        end

--)
                        --MOVX A,@Ri
                        -- acc <- ((r))
                        --
                      | (OPC_MOVX_1 as Imm7) then

                      begin

                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
--                    data_bus:=ram_in_data;
--assign to data_bus first got problem

                       xram_addr_tmp := (#ram_in_data @ {0,0,0,0,0,0,0,0} as Imm16);
                      START_RD_XRAM() ||
                      xram_in_data->ACC
                      -- cpu_state:=0b00
                        end


                        --MOVX A,@DPTR
                        -- acc <- (DPTR)
                        --
                      | (OPC_MOVX_2 as Imm7) then

                      begin
                       xram_addr_tmp := (#DPL @ #DPH as Imm16);
                      START_RD_XRAM() ||
                      xram_in_data->ACC
                      -- cpu_state:=0b00
                        end


                        --MOVX @Ri,A
                        -- ((r)) <- acc
                        --
                      | (OPC_MOVX_3 as Imm7) then

                      begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();

                       xram_addr_tmp := (#ram_in_data @ {0,0,0,0,0,0,0,0} as Imm16) ||
xram_out_data_tmp := reg_acc;
                      START_WR_XRAM()
                      -- cpu_state:=0b00
                        end


                        --MOVX @DPTR,A
                        -- (DPTR) <- acc
                        --
                      | (OPC_MOVX_4 as Imm7) then

                      begin
                       xram_addr_tmp := (#DPL @ #DPH as Imm16) || xram_out_data_tmp :=
reg_acc;
                      START_WR_XRAM()
                      -- cpu_state:=0b00
                        end
(--
                      --NOP
                        -- no operation
                        --
```

```
                      | (OPC_NOP as Imm7) then

                      begin
                      -- cpu_state:=0b00
                        end
--)
                        --ORL A,Rn
                        -- acc <- acc || (r)
                        --
                      | (OPC_ORL_1 as Imm7) then

                      begin
                      GET_RAM_ADDR_1();
                      START_RD_RAM(); ReadRAM();

                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2 := reg_acc; ALU();
                      ACC := (alu_des_1 as byte)
                      -- cpu_state:=0b00
                        end

                        --ORL A,direct
                        -- acc <- acc || (direct)
                        --
                      | (OPC_ORL_2 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();

                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2 := reg_acc; ALU();
                      ACC := (alu_des_1 as byte)
                      -- cpu_state:=0b00
                        end

                        --ORL A,@Ri
                        -- acc <- acc || ((r))
                        --
                      | (OPC_ORL_3 as Imm7) then

                      begin
                      GET_RAM_ADDR_2();
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;
                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();

                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1:=ram_in_data ||
alu_src_2 := reg_acc; ALU();
                      ACC := (alu_des_1 as byte)
                      -- cpu_state:=0b00
                        end

                        --ORL A,#data
                        -- acc <- acc || #data
                        --
                      | (OPC_ORL_4 as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1 := reg_acc  ||
alu_src_2 := reg_op2; ALU();
                      ACC := (alu_des_1 as byte)
                      -- cpu_state:=0b00
                        end

                        --ORL direct,A
                        -- (direct) <- (direct) || acc
                        --
                      | (OPC_ORL_5 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
```

```
                      START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1:=ram_in_data   ||
alu_src_2 := reg_acc; ALU();
                      ram_out_data_tmp :=(alu_des_1 as byte) || ram_addr_tmp
:=reg_op2;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --ORL direct,#data
                        -- (direct) <- (direct) || #data
                        --
                      | (OPC_ORL_6 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_OR as Imm4) || alu_src_1:=ram_in_data   ||
alu_src_2 := reg_op3; ALU();
                      ram_out_data_tmp :=(alu_des_1 as byte) || ram_addr_tmp
:=reg_op2;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --ORL C,bit
                        -- cy <- cy | (bit)
                        --
                      | (OPC_ORL_7 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_BIT_RAM(); ReadRAM();
                      data_bit:=ram_in_bit_data;
                      reg_cy := reg_cy or data_bit;
                      GET_PSW() -- cpu_state:=0b00
                        end

                        --ORL C,/bit
                        -- cy <- cy | ~(bit)
                        --
                      | (OPC_ORL_8 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_BIT_RAM(); ReadRAM();
                      data_bit:=ram_in_bit_data;
                      reg_cy := reg_cy or (not data_bit);
                      GET_PSW() -- cpu_state:=0b00
                        end

                        --POP direct
                        -- (direct) <- (sp)
                        -- sp <- sp - 1
                        --
                      | (OPC_POP as Imm7) then

                      begin
                      data_bus := SP;

                      alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=data_bus   ||
alu_src_2 := C0_8 ||
                      alu_src_cy := 0b1; ALU() || ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;

                      SP :=(alu_des_1 as byte);
                      ram_out_data_tmp :=data_bus || ram_addr_tmp :=reg_op2;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --PUSH direct
                        -- sp <- sp + 1
                        -- (sp) <- (direct)
                        --
```

```
                      | (OPC_PUSH as Imm7) then

                      begin
                      data_bus := SP;

                      alu_op_code:=(ALU_OPC_ADD as Imm4) || alu_src_1:=data_bus   ||
alu_src_2 := C0_8 ||
                      alu_src_cy := 0b1 ; ALU() ||
                       ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                      data_bus:=ram_in_data;

                      ram_out_data_tmp :=data_bus || ram_addr_tmp :=(alu_des_1 as
byte);
                      START_WR_RAM(); WriteRAM();
                      SP :=(alu_des_1 as byte)
                      -- cpu_state:=0b00
                        end


                        --RET
                        -- pc(15-8) <- (sp)
                        -- sp <- sp - 1
                        -- pc(7-0) <- (sp)
                        -- sp <- sp - 1
                        --
                      | (OPC_RET as Imm7) then

                      begin

                      data_bus := SP;
                      alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=data_bus ||
                      alu_src_2 := C0_8 || alu_src_cy := 0b1;
                      ALU() ||
                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                      reg_pch:=ram_in_data ;

                      ram_addr_tmp := (alu_des_1 as byte);
                      START_RD_RAM(); ReadRAM();
--
                      reg_pcl:=ram_in_data ||
--above OK
                      alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=(alu_des_1 as
byte) ||
                      alu_src_2 := C0_8 || alu_src_cy := 0b1;
ALU() ||
                      reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);
--above got problem

                      SP :=(alu_des_1 as byte);
                      SET_PC()
                      -- cpu_state:=0b00
;int_return <- 0b0

                        end

                        --RETI
                        -- pc(15-8) <- (sp)
                        -- sp <- sp - 1
                        -- pc(7-0) <- (sp)
                        -- sp <- sp - 1
                        --
                      | (OPC_RETI as Imm7) then

                      begin
                      int_return <- 0b1;
                      if (int_mask_int = 0b01) then int_lproc_int:=0b0 || int_lend<-
0b1
                      else   int_hproc_int:=0b0
                      end;
```

```
                      data_bus:=SP;
                      alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_1:=data_bus ||
                      alu_src_2 := C0_8 || alu_src_cy := 0b1; ALU() ||

                      ram_addr_tmp := data_bus;
                      START_RD_RAM(); ReadRAM();
                      reg_pch:=ram_in_data ;

                      ram_addr_tmp := (alu_des_1 as byte);
                      START_RD_RAM(); ReadRAM();
                      reg_pcl:=ram_in_data || alu_op_code:=(ALU_OPC_SUB as Imm4) ||
alu_src_1:=(alu_des_1 as byte)||
                      alu_src_2 := C0_8 || alu_src_cy := 0b1; ALU() ||
                      reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                      SP :=(alu_des_1 as byte);

                      SET_PC() ; -- cpu_state:=0b00;
                      int_return <- 0b0
                        end

                        --RL A
                        -- see I8051_ALU
                        --
                      | (OPC_RL as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_RL as Imm4) || alu_src_1:=reg_acc; ALU();
                      ACC :=(alu_des_1 as byte)  -- cpu_state:=0b00
                        end

                        --RLC A
                        -- see I8051_ALU
                        --
                      | (OPC_RLC as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_RLC as Imm4) || alu_src_1:=reg_acc ||
alu_src_cy:=reg_cy; ALU();
                      reg_cy:=alu_des_cy || ACC :=(alu_des_1 as byte);

                      GET_PSW() -- cpu_state:=0b00
                        end

                        --RR A
                        -- see I8051_ALU
                        --
                      | (OPC_RR as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_RR as Imm4) || alu_src_1:=reg_acc; ALU();
                      ACC :=(alu_des_1 as byte) -- cpu_state:=0b00
                        end

                        --RRC A
                        -- see I8051_ALU
                        --
                      | (OPC_RRC as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_RRC as Imm4) || alu_src_1:=reg_acc ||
alu_src_cy:=reg_cy; ALU();
                      reg_cy:=alu_des_cy || ACC :=(alu_des_1 as byte);
                      GET_PSW()  -- cpu_state:=0b00
                        end

                        --SETB C
                        -- cy <- 1
                        --
                      | (OPC_SETB_1 as Imm7) then

                      begin
```

```
                        reg_cy := 0b1;
                        GET_PSW()  -- cpu_state:=0b00
                          end

                          --SETB bit
                          -- (bit) <- 1
                          --
                        | (OPC_SETB_2 as Imm7) then

                        begin
                        ram_out_bit_data_tmp := 0b1 || ram_addr_tmp :=reg_op2;
                        START_WR_BIT_RAM(); WriteRAM() -- cpu_state:=0b00
                          end

                          --SJMP rel
                          -- pc <- pc + rel
                          --
                        | (OPC_SJMP as Imm7) then

                        begin
                        alu_op_code:=(ALU_OPC_PCSADD as Imm4) ||
alu_src_1:=(#reg_pc[0..7] as byte) ||
                        alu_src_2:=(#reg_pc[8..15] as byte) || alu_src_3:=reg_op2;
ALU();
                        reg_pcl:=(alu_des_1 as byte) || reg_pch:=alu_des_2;
                        reg_pc_8_10:=(#reg_pch[0..2] as Imm3) ||
reg_pc_11_15:=(#reg_pch[3..7] as Imm5);

                          SET_PC()
                        end

                          --SUBB A,Rn
                          -- acc <- acc - cy - (r)
                          --
                        | (OPC_SUBB_1 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc ||
                        alu_src_cy:=reg_cy; ALU();
                        ACC :=(alu_des_1 as byte) ||
                        reg_cy:=alu_des_cy || reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                        GET_PSW() -- cpu_state:=0b00
                          end

                          --SUBB A,direct
                          -- acc <- acc - cy - (direct)
                          --
                        | (OPC_SUBB_2 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc ||
                        alu_src_cy:=reg_cy; ALU();
                        ACC :=(alu_des_1 as byte) ||
                        reg_cy:=alu_des_cy || reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                        GET_PSW() -- cpu_state:=0b00
                          end

                          --SUBB A,@Ri
                          -- acc <- acc - cy - ((r))
                          --
                        | (OPC_SUBB_3 as Imm7) then

                        begin
                        GET_RAM_ADDR_2();
                        START_RD_RAM(); ReadRAM();
```

```
                        data_bus:=ram_in_data;
                        ram_addr_tmp := data_bus;
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc ||
                        alu_src_cy:=reg_cy; ALU();
                        ACC :=(alu_des_1 as byte) ||
                        reg_cy:=alu_des_cy || reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                        GET_PSW() -- cpu_state:=0b00
                         end

                         --SUBB A,#data
                         -- acc <- acc - cy - #data
                         --
                        | (OPC_SUBB_4 as Imm7) then

                        begin
                        alu_op_code:=(ALU_OPC_SUB as Imm4) || alu_src_2:=reg_op2 ||
alu_src_1:=reg_acc ||
                        alu_src_cy:=reg_cy; ALU();
                        ACC :=(alu_des_1 as byte) ||
                        reg_cy:=alu_des_cy || reg_ac:=alu_des_ac || reg_ov:=alu_des_ov;
                        GET_PSW() -- cpu_state:=0b00
                         end

                         --SWAP A
                         -- acc(3-0) <-> acc(7-4)
                         --
                        | (OPC_SWAP as Imm7) then

                        begin
                        ACC :=(#reg_acc[4..7] @ #reg_acc[0..3] as byte)--
cpu_state:=0b00
                         end

                         --XRL A,Rn
                         -- acc <-> (r)
                         --
                        | (OPC_XCH_1 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
                        ACC := data_bus;
                        ram_out_data_tmp := reg_acc;
                        START_WR_RAM(); WriteRAM()-- cpu_state:=0b00
                         end

                         --
                         -- acc <-> (direct)
                         --
                        | (OPC_XCH_2 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
                        ACC := data_bus;
                        ram_out_data_tmp := reg_acc;
                        START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                         end

                         --
                         -- acc <-> ((r))
                         --
                        | (OPC_XCH_3 as Imm7) then

                        begin
                        GET_RAM_ADDR_2();
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
```

```
                        ram_addr_tmp := data_bus;
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
                        ACC := data_bus;
                        START_WR_RAM(); WriteRAM();
                        ram_out_data_tmp := reg_acc;
                        START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                          end


                          --
                          -- acc(3-0) <-> ((r))(3-0)
                          --
                        | (OPC_XCHD as Imm7) then

                        begin
                        GET_RAM_ADDR_2();
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
                        ram_addr_tmp := data_bus;
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;

                        ACC :=(#data_bus[0..3] @ #reg_acc[4..7] as byte);
                        ram_out_data_tmp :=(#reg_acc[0..3] @ #data_bus[4..7] as byte);
                        START_WR_RAM(); WriteRAM()-- cpu_state:=0b00
                          end

                          --
                          -- acc <- acc ^ (r)
                          --
                        | (OPC_XRL_1 as Imm7) then

                        begin
                        GET_RAM_ADDR_1();
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc; ALU();
                        ACC :=(alu_des_1 as byte) || START_WR_RAM();
                        WriteRAM() -- cpu_state:=0b00
                          end

                          --
                          -- acc <- acc ^ (direct)
                          --
                        | (OPC_XRL_2 as Imm7) then

                        begin
                        ram_addr_tmp := reg_op2;
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc; ALU();
                        ACC :=(alu_des_1 as byte) || START_WR_RAM();
                        WriteRAM() -- cpu_state:=0b00
                          end

                          --
                          -- acc <- acc ^ ((r))
                          --
                        | (OPC_XRL_3 as Imm7) then

                        begin
                        GET_RAM_ADDR_2();
                        START_RD_RAM(); ReadRAM();
                        data_bus:=ram_in_data;
                        ram_addr_tmp := data_bus;
                        START_RD_RAM(); ReadRAM();

                        alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc; ALU();
                        ACC :=(alu_des_1 as byte)  || START_WR_RAM();
```

```
                      WriteRAM() -- cpu_state:=0b00
                        end


                        --
                        -- acc <- acc ^ #data
                        --
                      | (OPC_XRL_4 as Imm7) then

                      begin
                      alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=reg_op2 ||
alu_src_1:=reg_acc; ALU();
                      ACC :=(alu_des_1 as byte)  || START_WR_RAM();
                      WriteRAM()-- cpu_state:=0b00
                        end

                        --
                        -- (direct) <- (direct) ^ acc
                        --
                      | (OPC_XRL_5 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_acc; ALU();
                      ram_out_data_tmp :=(alu_des_1 as byte) || ram_addr_tmp
:=reg_op2;
                      START_WR_RAM(); WriteRAM() -- cpu_state:=0b00
                        end

                        --
                        -- (direct) <- (direct) ^ #data
                        --
                      | (OPC_XRL_6 as Imm7) then

                      begin
                      ram_addr_tmp := reg_op2;
                      START_RD_RAM(); ReadRAM();
                      alu_op_code:=(ALU_OPC_XOR as Imm4) || alu_src_2:=ram_in_data ||
alu_src_1:=reg_op3; ALU();
                      ram_out_data_tmp :=(alu_des_1 as byte) || ram_addr_tmp
:=reg_op2;
                      START_WR_RAM(); WriteRAM()
                      -- cpu_state:=0b00
                        end

                      else
                        begin
                              debug1<-0b11
                          end
-------------------------------------------------------------------------------
              end --end case op
end  -- end of exe


-------------------------------------------------------------------------------
shared Initialise is
begin

--reg_op1_tmp:=0b0 || reg_op2_tmp:=0b0 || reg_op3_tmp:=0b0 || fetch_op:=0b01 ||
-------------------------------------------------------------------------------
op_out_int:=(0 as array7) || reg_pc:=0 ||
int_hproc_int := 0b0 || int_lproc_int := 0b0 ||
P0_out:=(CM_8 as byte) || P1_out:=(CM_8 as byte) || P2_out:=(CM_8 as byte) || SP:=
0x07 || DPL:=0 || DPH:=0 || ACC:=0 ||
B:=0 || PSW:=0
|| tris0:=(CM_8 as byte) || tris1:=(CM_8 as byte) || tris2:=(CM_8 as byte)

;tris0_o <- tris0 ||
tris1_o <- tris1 || tris2_o <- tris2;
P0_o <- P0_out || P1_o <- P1_out || P2_o <- P2_out
end --end of initialise
```

```
shared DEC is

begin
        rmw_int:= 0b0;

        case (op_in_int) of
           ((ACALL as array5) @ {0bxxx} as byte)     then op_out_int :=(#(OPC_ACALL as
Imm7) as array7) || fetch_op:=0b10
        |  ((AJMP as array5) @ {0bxxx} as byte)      then op_out_int :=(#(OPC_AJMP as
Imm7) as array7) || fetch_op:=0b10
        |  (0b00101xxx)               then op_out_int :=(#(OPC_ADD_1 as Imm7) as
array7) || fetch_op:=0b00
        |  (0b00111xxx)        then op_out_int :=(#(OPC_ADDC_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b01011xxx)        then op_out_int :=(#(OPC_ANL_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b10111xxx)        then op_out_int :=(#(OPC_CJNE_3 as Imm7) as array7) ||
fetch_op:=0b11
        |  (0b11011xxx)        then op_out_int :=(#(OPC_DJNZ_1 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b00001xxx)        then op_out_int :=(#(OPC_INC_2 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b00011xxx)        then op_out_int :=(#(OPC_DEC_2 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b11101xxx)        then op_out_int :=(#(OPC_MOV_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b11111xxx)        then op_out_int :=(#(OPC_MOV_5 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b10101xxx)        then op_out_int :=(#(OPC_MOV_6 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b01111xxx)        then op_out_int :=(#(OPC_MOV_7 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b10001xxx)        then op_out_int :=(#(OPC_MOV_9 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b01001xxx)        then op_out_int :=(#(OPC_ORL_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b10011xxx)        then op_out_int :=(#(OPC_SUBB_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b11001xxx)        then op_out_int :=(#(OPC_XCH_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b01101xxx)        then op_out_int :=(#(OPC_XRL_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (ADD_2 as byte)     then op_out_int :=(#(OPC_ADD_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ADD_4 as byte)     then op_out_int :=(#(OPC_ADD_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ADDC_2 as byte)    then op_out_int :=(#(OPC_ADDC_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ADDC_4 as byte)    then op_out_int :=(#(OPC_ADDC_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ANL_2 as byte)     then op_out_int :=(#(OPC_ANL_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ANL_4 as byte)     then op_out_int :=(#(OPC_ANL_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ANL_5 as byte)     then op_out_int :=(#(OPC_ANL_5 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (ANL_6 as byte)     then op_out_int :=(#(OPC_ANL_6 as Imm7) as array7) ||
fetch_op:=0b11 || rmw_int := 0b1
        |  (ANL_7 as byte)     then op_out_int :=(#(OPC_ANL_7 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ANL_8 as byte)     then op_out_int :=(#(OPC_ANL_8 as Imm7) as array7) ||
fetch_op:=0b10
        |  (CJNE_1 as byte)    then op_out_int :=(#(OPC_CJNE_1 as Imm7) as array7) ||
fetch_op:=0b11
        |  (CJNE_2 as byte)    then op_out_int :=(#(OPC_CJNE_2 as Imm7) as array7) ||
fetch_op:=0b11
        |  (CLR_1 as byte)     then op_out_int :=(#(OPC_CLR_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (CLR_2 as byte)     then op_out_int :=(#(OPC_CLR_2 as Imm7) as array7) ||
fetch_op:=0b00
```

```
        |  (CLR_3 as byte)     then op_out_int :=(#(OPC_CLR_3 as Imm7) as array7) ||
fetch_op:=0b10
        |  (CPL_1 as byte)     then op_out_int :=(#(OPC_CPL_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (CPL_2 as byte)     then op_out_int :=(#(OPC_CPL_2 as Imm7) as array7) ||
fetch_op:=0b00
        |  (CPL_3 as byte)     then op_out_int :=(#(OPC_CPL_3 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (DA as byte)               then op_out_int :=(#(OPC_DA as Imm7) as array7)
|| fetch_op:=0b00
        |  (DEC_1 as byte)     then op_out_int :=(#(OPC_DEC_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (DEC_3 as byte)     then op_out_int :=(#(OPC_DEC_3 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (DIV as byte)       then op_out_int :=(#(OPC_DIV as Imm7) as array7) ||
fetch_op:=0b00
        |  (DJNZ_2 as byte)    then op_out_int :=(#(OPC_DJNZ_2 as Imm7) as array7) ||
fetch_op:=0b11 || rmw_int := 0b1
        |  (INC_1 as byte)     then op_out_int :=(#(OPC_INC_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (INC_3 as byte)     then op_out_int :=(#(OPC_INC_3 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (INC_5 as byte)     then op_out_int :=(#(OPC_INC_5 as Imm7) as array7) ||
fetch_op:=0b00
        |  (JB as byte)               then op_out_int :=(#(OPC_JB as Imm7) as array7)
|| fetch_op:=0b11
        |  (JBC as byte)       then op_out_int :=(#(OPC_JBC as Imm7) as array7) ||
fetch_op:=0b11
        |  (JC as byte)               then op_out_int :=(#(OPC_JC as Imm7) as array7)
|| fetch_op:=0b10
        |  (JMP as byte)       then op_out_int :=(#(OPC_JMP as Imm7) as array7) ||
fetch_op:=0b00
        |  (JNB as byte)       then op_out_int :=(#(OPC_JNB as Imm7) as array7) ||
fetch_op:=0b11
        |  (JNC as byte)       then op_out_int :=(#(OPC_JNC as Imm7) as array7) ||
fetch_op:=0b10
        |  (JNZ as byte)       then op_out_int :=(#(OPC_JNZ as Imm7) as array7) ||
fetch_op:=0b10
        |  (JZ as byte)               then op_out_int :=(#(OPC_JZ as Imm7) as array7)
|| fetch_op:=0b10
        |  (LCALL as byte)     then op_out_int :=(#(OPC_LCALL as Imm7) as array7) ||
fetch_op:=0b11
        |  (LJMP as byte)      then op_out_int :=(#(OPC_LJMP as Imm7) as array7) ||
fetch_op:=0b11
        |  (MOV_2 as byte)     then op_out_int :=(#(OPC_MOV_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOV_8 as byte)     then op_out_int :=(#(OPC_MOV_8 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOV_10 as byte)    then op_out_int :=(#(OPC_MOV_10 as Imm7) as array7) ||
fetch_op:=0b11
        |  (MOV_12 as byte)    then op_out_int :=(#(OPC_MOV_12 as Imm7) as array7) ||
fetch_op:=0b11
        |  (MOV_16 as byte)    then op_out_int :=(#(OPC_MOV_16 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOV_17 as byte)    then op_out_int :=(#(OPC_MOV_17 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOV_18 as byte)    then op_out_int :=(#(OPC_MOV_18 as Imm7) as array7) ||
fetch_op:=0b11
        |  (MOVC_1 as byte)    then op_out_int :=(#(OPC_MOVC_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (MOVC_2 as byte)    then op_out_int :=(#(OPC_MOVC_2 as Imm7) as array7) ||
fetch_op:=0b00
        |  (ORL_4 as byte)     then op_out_int :=(#(OPC_ORL_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ORL_5 as byte)     then op_out_int :=(#(OPC_ORL_5 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (ORL_6 as byte)     then op_out_int :=(#(OPC_ORL_6 as Imm7) as array7) ||
fetch_op:=0b11 || rmw_int := 0b1
        |  (ORL_7 as byte)     then op_out_int :=(#(OPC_ORL_7 as Imm7) as array7) ||
fetch_op:=0b10
        |  (ORL_8 as byte)     then op_out_int :=(#(OPC_ORL_8 as Imm7) as array7) ||
fetch_op:=0b10
```

```
        |  (POP as byte)       then op_out_int :=(#(OPC_POP as Imm7) as array7) ||
fetch_op:=0b10
        |  (PUSH as byte)      then op_out_int :=(#(OPC_PUSH as Imm7) as array7) ||
fetch_op:=0b10
        |  (RET as byte)       then op_out_int :=(#(OPC_RET as Imm7) as array7) ||
fetch_op:=0b00
        |  (RETI as byte)      then op_out_int :=(#(OPC_RETI as Imm7) as array7) ||
fetch_op:=0b00
        |  (RL as byte)               then op_out_int :=(#(OPC_RL as Imm7) as array7)
|| fetch_op:=0b00
        |  (RLC as byte)       then op_out_int :=(#(OPC_RLC as Imm7) as array7) ||
fetch_op:=0b00
        |  (RR as byte)               then op_out_int :=(#(OPC_RR as Imm7) as array7)
|| fetch_op:=0b00
        |  (RRC as byte)       then op_out_int :=(#(OPC_RRC as Imm7) as array7) ||
fetch_op:=0b00
        |  (SETB_1 as byte)    then op_out_int :=(#(OPC_SETB_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (SETB_2 as byte)    then op_out_int :=(#(OPC_SETB_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (SJMP as byte)      then op_out_int :=(#(OPC_SJMP as Imm7) as array7) ||
fetch_op:=0b10
        |  (SUBB_2 as byte)    then op_out_int :=(#(OPC_SUBB_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOVX_2 as byte)    then op_out_int :=(#(OPC_MOVX_2 as Imm7) as array7) ||
fetch_op:=0b00
        |  (MOVX_4 as byte)    then op_out_int :=(#(OPC_MOVX_4 as Imm7) as array7) ||
fetch_op:=0b00
        |  (MUL  as byte)      then op_out_int :=(#(OPC_MUL as Imm7) as array7) ||
fetch_op:=0b00
        |  (NOP  as byte)      then op_out_int :=(#(OPC_NOP as Imm7) as array7) ||
fetch_op:=0b00
        |  (ORL_2 as byte)     then op_out_int :=(#(OPC_ORL_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (SUBB_4 as byte)    then op_out_int :=(#(OPC_SUBB_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (SWAP as byte)      then op_out_int :=(#(OPC_SWAP as Imm7) as array7) ||
fetch_op:=0b00
        |  (XCH_2 as byte)     then op_out_int :=(#(OPC_XCH_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (XRL_2 as byte)     then op_out_int :=(#(OPC_XRL_2 as Imm7) as array7) ||
fetch_op:=0b10
        |  (MOV_4 as byte)     then op_out_int :=(#(OPC_MOV_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (XRL_4 as byte)     then op_out_int :=(#(OPC_XRL_4 as Imm7) as array7) ||
fetch_op:=0b10
        |  (XRL_5 as byte)     then op_out_int :=(#(OPC_XRL_5 as Imm7) as array7) ||
fetch_op:=0b10 || rmw_int := 0b1
        |  (XRL_6 as byte)     then op_out_int :=(#(OPC_XRL_6 as Imm7) as array7) ||
fetch_op:=0b11 || rmw_int := 0b1
        |  (0b0010011x)        then op_out_int :=(#(OPC_ADD_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b0011011x)        then op_out_int :=(#(OPC_ADDC_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b0101011x)        then op_out_int :=(#(OPC_ANL_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1011011x)        then op_out_int :=(#(OPC_CJNE_4 as Imm7) as array7) ||
fetch_op:=0b11
        |  (0b0001011x)        then op_out_int :=(#(OPC_DEC_4 as Imm7) as array7) ||
fetch_op:=0b00 || rmw_int := 0b1
        |  (0b0000011x)               then op_out_int :=(#(OPC_INC_4 as Imm7) as
array7) || fetch_op:=0b00 || rmw_int := 0b1
        |  (0b1110011x)        then op_out_int :=(#(OPC_MOV_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1000011x)        then op_out_int :=(#(OPC_MOV_11 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b1111011x)        then op_out_int :=(#(OPC_MOV_13 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1010011x)        then op_out_int :=(#(OPC_MOV_14 as Imm7) as array7) ||
fetch_op:=0b10
        |  (0b0111011x)        then op_out_int :=(#(OPC_MOV_15 as Imm7) as array7) ||
fetch_op:=0b10
```

```
        |  (0b1110001x)        then op_out_int :=(#(OPC_MOVX_1 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1111001x)        then op_out_int :=(#(OPC_MOVX_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b0100011x)        then op_out_int :=(#(OPC_ORL_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1001011x)        then op_out_int :=(#(OPC_SUBB_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1100011x)        then op_out_int :=(#(OPC_XCH_3 as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b1101011x)        then op_out_int :=(#(OPC_XCHD as Imm7) as array7) ||
fetch_op:=0b00
        |  (0b0110011x)        then op_out_int :=(#(OPC_XRL_3 as Imm7) as array7) ||
fetch_op:=0b00

        else op_out_int := (#(OPC_ERROR as Imm7) as array7) || fetch_op:=0b00
         end --end if
end --end X8051_DEC

shared Fetch is
begin

                      --debug1<-0b01 ||
                      pc_out <- reg_pc;
                      -------------------------
                      InstrFetch() || PcInc();
                      reg_op1:=reg_ir || op_in_int := reg_ir || reg_pc:=reg_pc_plus;
-------------------------------------------------------------------------------------
-----
---- fetch_op inside DEC causes the 2nd read to come much later-----------------------
-----
                      DEC();
                        X8051_Fetch()
                      --below for testing only
                      ;acc_out <- reg_acc || psw_out <- reg_psw
                      ------------------------
                      ||
                      -- below for testing only
                      debug1<-0b10
end



-------------------------------------------------------------------------------
--------------------------main loop ---------------------------------------------
begin
Initialise();

loop
---- cpu_state=0
                      -- below for testing only
--                    debug1<-0b00;
                      -------------------------
                      Interrupt();
                      Fetch();
                      EXE()
end
end --end main
```