"SoC Level Functional verification

And Application Design of a Smart 3-Axis Accelerometer Chip"

Major Project Report

Submitted In Partial Fulfillment of the Requirement

For

MASTER OF TECHNOLOGY

in

ELECTRONICS & COMMUNICATION ENGG.

(VLSI DESIGN)

By

Kirit V. Patel(07MEC013)

External Project Leader :

Mr. Mohammad Haris Minai

Freescale Semiconductor Pvt. Ltd,

Noida.

Internal Project Guide :

Prof. N. P. Gajjar

EC Department ,

Institute of Technology,

Nirma University, Ahmedabad

Department of Electronics & Communication Engineering

Institute of Technology,

Nirma University of Science And Technology

AHMEDABAD-382481

Certificate

This is to certify that the Major Project entitled "SoC Level Functional verification And Application Design of a Smart 3-Axis Accelerometer Chip" submitted by Kirit V. Patel (07MEC013), towards the partial fulfillment of the requirements for the degree of Master of Technology in Electronic & Communicaiton of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Project Guide:
Prof. N.P. Gajjar
Department of EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

P.G. Co-ordinator
Dr. N. M. Devashrayee
Department of EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

HOD (E.C. Dept.):
Prof. A. S. Ranade,
Department of EC Engineering,
Institute of Technology,
Nirma University,Ahmedabad

Director:
Dr K Kotecha
Department of EC Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

An accelerometer is a device for measuring acceleration and gravity induced reaction forces. From this measurement, any device can run some smart application. Single- and multi-axis models are available to detect magnitude and direction of the acceleration.

- Design ,programming and analysis of Application

Accelerometer is programmed for the following detection and it's purpose. I have generated algorithm for following application and also programmed in coldfire language. In some application I have design without coldfire processor and with coldfire processor, then compaires the memory and time requirement for running the application.

1. 0g (fall) detection for preventation of data loss.

Important of fall detection:-

A key benefit of laptop computers is that they are mobile, but with that mobility comes the risk of being dropped. High shock events put the hard- disk drive and the user data contained in it at risk. With more users transitioning from desktops to laptops as their primary computers, it has become increasingly important to provide a robust solution to help protect the hard-disk drive and prevent data loss. So accelerometer will detect the free fall detection and power off the hard disk so data will not be lost.

2.Tilt and 3D orientation detection for resolution improvement

3.Tap and double tap detection for run or stop any application

4.Dead reckoning

5.Shock, vibration and sudden motion detection

I have done the analysis of above application related to coldfire CPU. It also shows the information regarding memory size , total number of cycle requirement and power consumption for a particular application. It also shows the memory size, total cycle, power consumption with respect to different sampling rate.

- SOC level verification

The accelerometer include number of blocks , such as Modulo timer, Power delay block, Port controller, clk generator ,system integration module, memory, etc. Among them some blocks are used in system level application So I have generated system level testcases for the Power delay block , port control and Modulo timer for the SOC level verification. These testcases are in c language, verilog nd system verilog .
This Accelerometer is used in below areas for the smart application:

- Cellophanes.
- Personal Navation Devices.
- Pedometry.
- Gaming and Toys.

Acknowledgements

I express my gratitude and appreciation for all those with whom I worked and interacted at Freescale Semiconductor Pvt. Ltd , Noida and at Institute of Technology, Nirma University, Ahmedabad, and thank all of them for their help and co-operation.

First and foremost I would like to express my heartily gratitude to Prof. N.P.Gajjar Institute of Technology, Nirma University, Mr. Vivek Goel , Mr. Mohammad Haris Minai And Mr. Nandan Tripathi Freescale Semiconductor Pvt. Ltd , Noida for giving me the permission and providing the facilities for this project.

I am also thankful to Dr. N. M. Devashrayee and Prof. Usha Mehta and Prof. Amisha P. Naik for providing me the able guidance to carried out the project work. I also gratefully acknowledge Mrs. Neeti B. Avsatthi for providing me the full laboratory support at PG- VLSI Design Lab.
What and whare I am today is due to my parent's love and constant encouragement throughout my life. I would like to dedicate my work to my parents , Vishnubhai G. Patel and Sitaben v. Patel.

Finally, I would like to thank entire staff of EC Department, Institute of Technology, Nirma University, Ahmedabad.

Kirit Patel (07MEC013)
M.Tech (VLSI Design)
Institute of Technology,
Nirma University of Science and Technology, Ahmedabad

Contents

Certificate **ii**

Abstract **iii**

Acknowledgements **v**

List of Figures **1**

1 Introduction **2**
1.1 General Description 2
1.2 Hardware Features : 3
1.3 Software Features : 4
1.4 Typical Applications 4

2 Block Diagram **6**

3 Coldfire Processor **9**
3.1 ColdFire Programming Model 9
3.1.1 Data Registers (D0-D7) 9
3.1.2 Address Registers (A0-A7) 9
3.1.3 Program Counter (PC) 10
3.1.4 Condition Code Register (CCR) 10

4 Software Application **12**
4.1 Freefall detection 12
4.2 Single and Double Click (Tap) Detection 15
4.2.1 4.2.1 Single click 15
4.2.2 Double click 16
4.3 Slope Detection 17
4.4 Swing Detection Algorithm 17
4.5 Turnover Detection Algorithm 18
4.6 Rolling Dice Detection Algorithm 19

5 Tap /Double Tap Detection at system Level **21**
5.1 Pseudo Code 21
5.2 Execute Path 23

6 Flip detection At System Level **27**

7 A Typical SOC Device **30**

8 SoC Verification Environment **33**
8.1 Verification challenges And Solution 33
8.2 Traditional SOC Verification 34
8.3 Verification Planning Guidelines 36
8.4 SoC Verification Flow 37

9 Modulo Timer **44**
9.1 Introduction 44
9.2 Stimulus 46
9.2.1 Stimulus Organization 46
9.2.2 Stimulus Template 47
9.2.3 Stimulus Directory Structure 47
9.3 Coding for Verification 47

10 Programmable Delay Block **52**

11 Summary **56**

References **59**

Index **59**

List of Figures

2.1 Block Diagram for Digital Communication System 8

3.1 ColdFire Programming Model 10
3.2 structure 11

4.1 Acceleration Detection 14
4.2 : Single click event with non latched interrupt 17
4.3 Single and double click recognition 18
4.4 slope detection 19
4.5 swing detection 20

6.1 flip Detection 28

7.1 SOC Device 32

8.1 SoC verification flow 38

9.1 Timer Module Connection 45
9.2 Testbench Architecture 48

10.1 Block Diagram 53
10.2 Programmable Delay Block Control and Status Register (CSR) . . . 54
10.3 PDB connection 55

Chapter 1

Introduction

1.1 General Description

An accelerometer is a device for measuring acceleration and gravity induced reaction forces. Single- and multi-axis models are available to detect magnitude and direction of the acceleration.

Accelerometer is proposed as a smart digital 3-axis low-g accelerometer. The main components of the accelerometer platform are a 3-axis MEMS g-sensor and an ASIC containing a CPU, memories, ADCs and supporting peripherals. The accelerometer related applications can be offloaded from the main CPU to the CPU present on accelerometer.

Some typical tasks that the accelerometer needs to take care of are: 0g detection, tilt and 3D orientation detection, tap and double tap detection, dead reckoning, pedometery, shock, vibration and sudden motion detection etc. The basic algorithms for these applications along with corresponding Ccode and assembly codes for Coldfire V1 CPUs are used to arrive at the MIPS , memory requirement and power requirement for the accelerometer CPU.

This low-G acceleromter is a member of Freescale's family of digital readout accelerometers. This device incorporates dedicated MEMS transducers, signal conditioning, data conversion and 32-bit programmable CPU for digital signal processing.

1.2 Hardware Features :

Three accelerometer operating ranges.

- +/- 2g suits most user interaction (mouse) motions and freefall.
- +/- 4g covers most regular human dynamics (walking, jogging, etc.)
- +/- 8g detects most abrupt activities (toys)
- Integrated temperature sensor
- One Slave SPI or I2C interface operates up to 2MBPS
- One Master I2C interface operates up to 400KBPSmm
- 10 and 12 bit data formats available o 1.8 V supply voltage
- 32-bit ColdFire V1 CPU
- Extensive set of power management features and low power modes.
- Integrated 14-bit ADC
- Single wire Background Debug Mode (BDM) pin interface
- 16KB Flash Memory o 2K Random Access Memory
- Two channel timer with input capture, output capture or edge-aligned PWM

- Programmable delay block for scheduling events relative to start of frame
- Modulo timer for scheduling periodic events
- Minimal external component requirements

1.3 Software Features :

This device may be programmed to provide any of the following:

- 0g (fall) detection
- Tilt and 3D orientation detection
- tap and double tap detection
- Dead reckoning
- Shock, vibration and sudden motion detection
- Power management

1.4 Typical Applications

This intelligent sensor is optimized for use in low voltage, portable, consumer products such as:

- Cellphones
- Personal Navation Devices (PNDs)
- Pedometry
- Gaming and Toys.

This accelerometer can manage a secondary sensor, such as a pressure sensor or magnetometer, allowing the main application processor to be powered down until absolutely needed.

Chapter 2

Block Diagram

The accelerometer family is a satellite accelerometer which is comprised of a three axis MEMS accelerometer and interface IC. The interface IC converts the analog signal to a digital format, which can then be processed using the on-chip 32-bit CPU. The digital value is then accessible to the system master via the slave I2C port.

Accelerometer can also process up to three external analog signals, allowing it to act as controller for additional sensing devices such as pressure sensors and magnetometers.

A high level view of accelerometer is shown in Figure . Key components include:

- o The 3-axis transducer is shown. This block is entirely passive, and includes the MEMS structures.

  The AFE, or Analog Front End, is composed of:

- capacitance to voltage converter (C2V)
- analog to digital converter
- temperature sensor

The digital sub-system, composed of :

- 32-bit ColdFire V1 CPU
- Memory: RAM, ROM and Flash
- RGPIO port control logic
- Timer Functions
- Modulo (MTIM16) Programmable delay timer (PDB)
- General purpose input/output capture (TPM)
- System Integration Module (SIM)
- I2C master interface
- Clock generation module

Excluding the clock generation function, the processing sub-system is generally shut down whenever the Analog Front End is active, and vice-versa. This is done to minimize noise impacts on the AFE. The slave interfaces (one of SPI or I2C) operate independently of the CPU subsystem. They can be accessed at any time.

- The three axis sensor will sense the acceleration. This sensor are MEMS based ,so the output is in the form of capacitance.
- The capacitive output of the sensor is converted in the form of voltage using C to V converter.
- Coldfire input is in the form of digital. So input voltage is converted in the form of digital using ADC converter. This ADC is 14 bit .
- Coldfire CPU perform the process on the digital data and run the software application as per the user requirement
- Mux select the one axis from the three axis at a time.

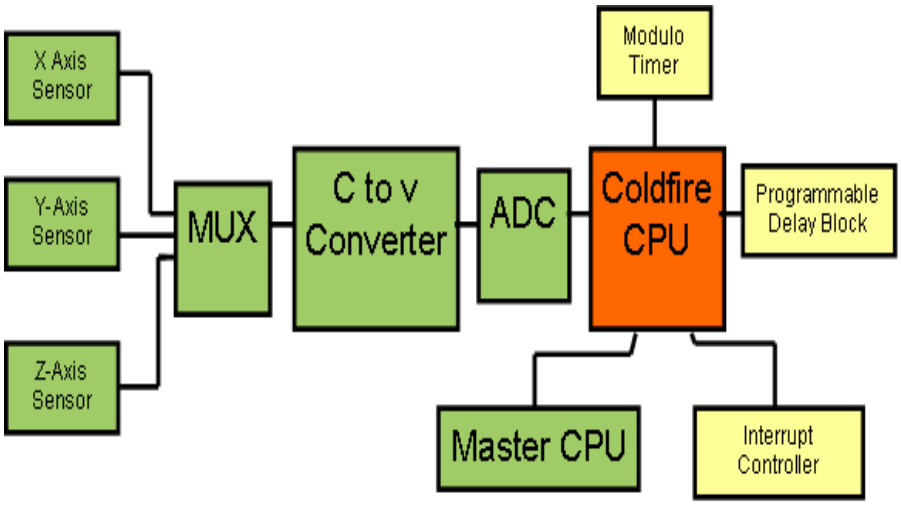

Figure 2.1: Block Diagram for Digital Communication System

??

Chapter 3

Coldfire Processor

This section describes the organization of the Version ColdFire processor core and an overview of the program-visible registers

3.1 ColdFire Programming Model

It consists 16 general-purpose 32-bit registers (D0-D7, A0-A7) , 32-bit program counter (PC) and 8-bit condition code register (CCR)

3.1.1 Data Registers (D0-D7)

These registers are for bit, byte (8 bits), word (16 bits), and longword (32 bits) operations. They can also be used as index registers

3.1.2 Address Registers (A0-A7)

These registers serve as software stack pointers, index registers, or base address registers. The base address registers can be used for word and longword operations. Register A7 functions as a hardware stack pointer during stacking for subroutine calls and exception handling.

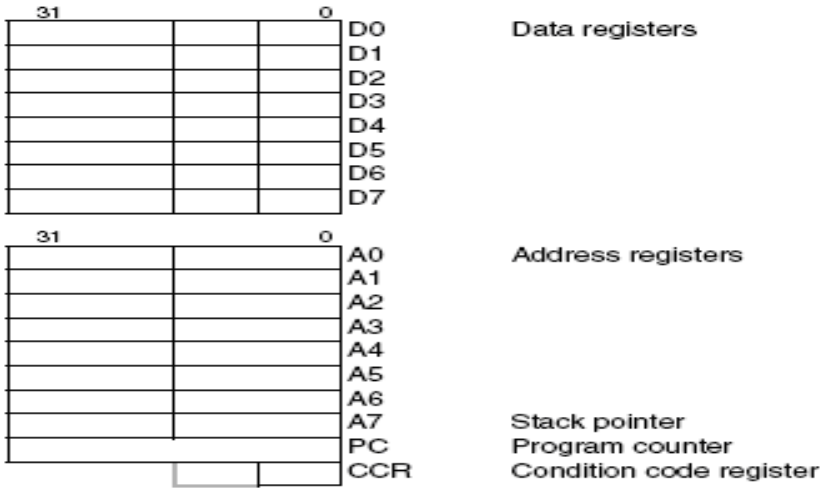

Figure 3.1: ColdFire Programming Model

3.1.3 Program Counter (PC)

The program counter (PC) contains the address of the instruction currently executing. During instruction execution and exception processing, the processor automatically increments the contents or places a new value in the PC. For some addressing modes, the PC can serve as a pointer for PC relative addressing.

3.1.4 Condition Code Register (CCR)

Consisting of 5 bits, the condition code register (CCR)-the status register's lower byte- is the only portion of the SR available in the user mode. Many integer instructions affect the CCR and indicate

- Bit : X Extend: Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.
- Bit : N Negative: Set if the most significant bit of the result is set; otherwise cleared

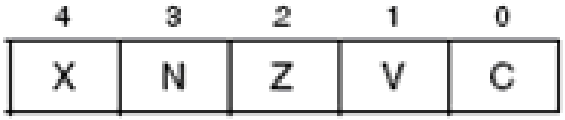

Figure 3.2: structure

- Bit : Z Zero :Set if the result equals zero; otherwise cleared.
- Bit :V Overflow: Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared.
- Bit :C Carry. Set if a carry out of the most significant bit of the operand occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared.

Chapter 4

Software Application

On the bases of coldfire processor and the analog system, we can run the software application in our mobile. The application lists are shown below:

- Freefall detection
- Single and Double Click (Tap) Detection
- Slope Detection
- Swing Detection Algorithm
- Turnover Detection Algorithm
- Rolling Dice Detection Algorithm

4.1 Freefall detection

A key benefit of laptop computers is that they are mobile, but with that mobility comes the risk of being dropped. High shock events put the hard-disk drive and the user data contained in it at risk. With more users transitioning from desktops to laptops as their primary computers, it has become increasingly important to provide

a robust solution to help protect the hard-disk drive and prevent data loss

To meet this need, It offers drop (or "free-fall") protection as a standard feature in its laptop computers equipped with 7200 revolutions per minute (RPM) hard-disk drives. This feature is designed to detect a fall and protect the hard-disk drive by parking its heads before impact. The new Hard Disk Drive with Free Fall Sensor is offered.

Free-fall protection implementations vary significantly in how quickly they respond after the laptop is dropped. The response time is governed largely by the location of the free-fall detection and response mechanisms. Traditional implementations locate this mechanism on the system board, which introduces latency due to system overhead. In contrast, the innovative new 7200- RPM hard drives in It systems locate the detection and response mechanisms directly in the hard drive itself.

This approach eliminates system overhead and yields significantly better response time-translating to a more expansive "protected zone," as shown in Figure.

- How Does Free-Fall Protection Work?

With the exception of fans, the hard-disk and optical- disk drives are generally the only mechanical moving parts of a computer. The design of the hard-disk drive mimics that of a record player, with an actuator arm seeking, reading, and writing information in the form of bits (1s and 0s) on magnetic media. The hard-disk drive has one or more sensitive magnetic heads that, like the needle of a record player, travel over the circular media to store and retrieve data. To adequately protect the head and media during a shock event and avoid the resulting loss of data, the head(s) must be rapidly moved away from the media and "parked" in a safe location. The main challenges associated with this process are to reliably detect free-fall motion and then park the heads prior to the point at which the impact occurs. Because of the motions involved in the typical uses of laptop computers-such as typing, walking,

or closing the lid-sensing free-fall events involves a complex detection process.

To minimize false detections, the process must discriminate between the normal operation of the laptop and actual free-fall events. In addition, once a free-fall event is detected, the system must park the hard-disk drive head(s) rapidly before the laptop experiences the collision.

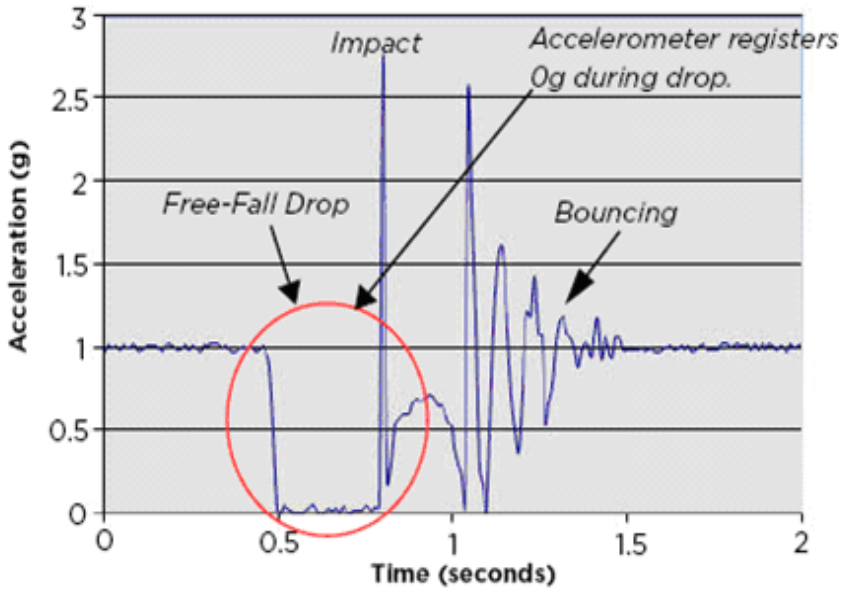

Figure 4.1: Acceleration Detection

- Explanation

In case of a 3-axis accelerometer, when the device is stationary the total magnitude of the acceleration on the sensor should be equal to 1g (-9.81 m/s2).

The accelerometer outputs in case of a linear fall. The window region detection hardware/software provides an indication (e.g. through an interrupt) whenever an accelerometer output crosses a predefined threshold. This will invoke a software

routine. A freefall condition is defined as: (Ax ¡ threshold) and (Ay ¡ threshold) and (Az ¡ threshold)

To avoid faulty detection due to noise or a glitch, the algorithm may chose to wait for more than one sample to make a decision of a free fall (averaging filter). Butthis comes at a price of longer time in making a decision and hence a more distance covered in fall. Use of basic equation of motion can easily give a comparative idea for this.

4.2 Single and Double Click (Tap) Detection

Theory of operation The single click and double click recognition functions featured i help to create a man-machine interface with little software loading. The device can be configured to output an interrupt signal on a dedicated pin when tapped in any direction. If the sensor is exposed to a single input stimulus, it generates an interrupt request on inertial interrupt pin INT1 and/or INT2. A more advanced feature allows the generation of an interrupt request when a double input stimulus with programmable time between the two events is recognized, enabling a mouse button-like functionality. This function can be fully programmed by the user in terms of expected amplitude and timing of the stimuli by means of the dedicated set of registers The single and double click recognition works independently on the selected output data rate

4.2.1 4.2.1 Single click

If the device is configured for single click event detection, an interrupt is generated when the input acceleration on the selected channel exceeds the programmed threshold, and returns below it within a time window defined by the TimeLimit register. If the LIR bit of the CLICK CFG register is not set, the interrupt is kept high for the duration of the Latency window. If the LIR bit is set, the interrupt is kept high until the CLICK SRC register is read.

In Figure (a) the click has been recognized, while in Figure (b) the click has not been recognized because the acceleration goes under the threshold after the TimeLimit has expired

4.2.2 Double click

If the device is configured for double click event detection, an interrupt is generated when, after a first click, a second click is recognized. The recognition of the second click occurs only if the event satisfies the rules defined by the Latency and Windows registers. In particular, after the first click has been recognized, the second click detection procedure is delayed for an interval defined by the Latency register. This means that after the first click has been recognized, the second click detection procedure will start only if the input acceleration exceeds the threshold after the Latency window but before the Window has expired [Figure (a)] or if the acceleration is still above the threshold after the Latency has expired [Figure (b)].

Once the second click detection procedure is initiated, the second click will be recognized with the same rule as the first: the acceleration must return below the threshold before the TimeLimit has expired.
Appropriately defining the Latency window is important to avoid unwanted clicks due to spurious bouncing of the input signal.

Figure illustrates a single click event (a) and a double click event (b). The device is able to distinguish between (a) and (b) by changing the settings of the CLICK CFG register from single to double click recognition.

In Figure (a) the double click event has been correctly recognized, while in Figure (b) the interrupt has not been generated because the input acceleration exceeds the threshold after the Window interval has expired.

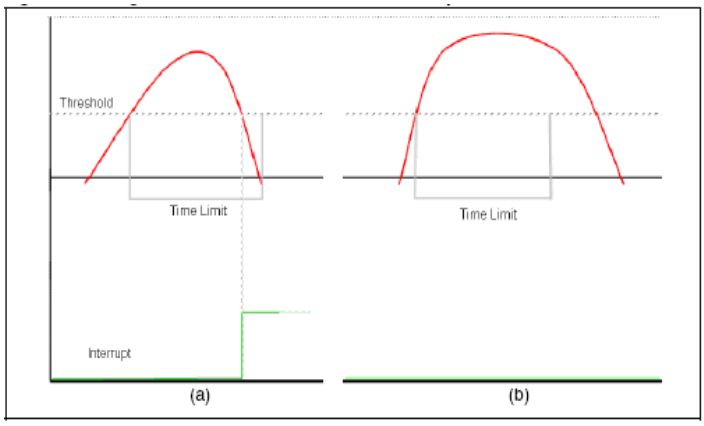

Figure 4.2: : Single click event with non latched interrupt

4.3 Slope Detection

- The sensed object is turned left or right about Y axis, as shown as the blue arrow on figure , the slope position is detected and reported, so that the application software could respond and control the application accordingly.

- The slope detection algorithm is designed to detect the movement of the object from flat position to left slope, or to right slope.

4.4 Swing Detection Algorithm

- This algorithm is designed to detect the posture of the sensed object. The algorithm is based on an assumption that the central or original position of the object is like that the traverse axis of the object is flat and the lengthways axis of the object is about 40 degrees to the horizontal surface, just like the position

- The cell phone shown on figure 1, because this position is the most central position of a portable devices held by a person's hands.

- When the sensed object swings to either of the 8 directions (up, down, right,

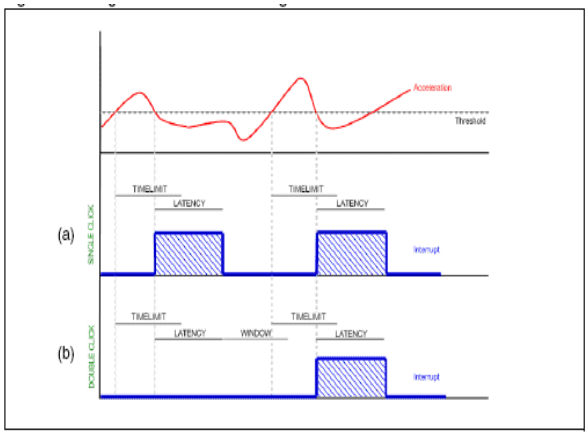

Figure 4.3: Single and double click recognition

left, up-right, up-left, down-right and down-left), as shown as the red arrows on figure 1, and as the swing amplitude is large enough, a swing movement is detected and reported, so that the application software could respond and control the application accordingly

4.5 Turnover Detection Algorithm

- This algorithm is designed to detect the turnover movement of an object.
- When the object is turned over from an original position to the opposite position, a turnover movement is detected.

Example:

- An cellphone, when it's turned from the screen-up position to screedown position, or vice versa.
- The positions between which a turnover movement is detected are usually at the direction of one axis, which is always Z axis of the accelerometer

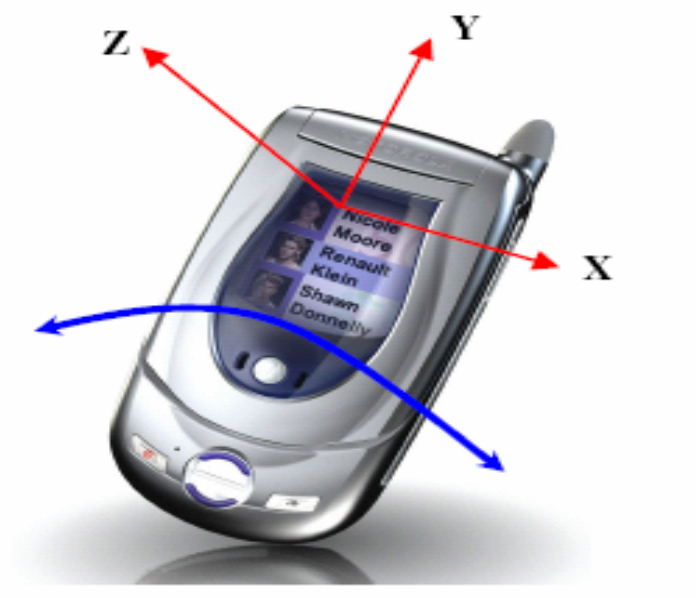

Figure 4.4: slope detection

4.6 Rolling Dice Detection Algorithm

- To implement a game of rolling dice.
- On this algorithm the force's (or acceleration) strength, direction and lasting time applied on an object are calculated before the final status of dices is concluded.
- The final status means how long the dices should keep rolling after the object is stop moving, and when rolling stops, which number is on each dice's face. The number of dices can be chosen by users, from 1 to 6.

This mode selects the algorithm for six dice2 rolling together based on the sensor movement. Sensor movement in directions X, Y, and Z will flip the die face accordingly. This algorithm p resumes six dice rolling at the same time. Each die experiences a slightly different rotational force, such that the rotational results of the dice are different over time.

Figure 4.5: swing detection

When this mode is entered, the low-G sensor is enabled. It continuously monitors the movement of the unit. When a dramatic change in sensor movement is detected, the dice rolling algorithm starts and interrupt pin INT is asserted. When the sensor movement is stopped, the dice rolling algorithm will continue to run for a a few seconds and then stop. Interrupt pin INT is asserted again.

The algorithm stopping time depends on the time taken to roll the dice. The longer it is, the more delay there is before the algorithm stops. The longest delay is about four seconds. The corresponding interrupt flag can be read by command FW Status to understand the source of the interrupt. The interrupt condition is cleared by the command Int Ack. The rolling dice result can be read by command Dice Read1, Dice Read2, Dice Read3, Dice Read4, Dice Read5, and Dice Read6, according to the die number to be read.

Chapter 5

Tap /Double Tap Detection at system Level

If the device is enabled for tap/double tap detect application, sensed value is used to determine the tap occurrence. Tap is said to have occurred if any of the sensor output crosses a predefined threshold and returns back to the normal value within a predefined time interval Ttap interval . A double tap event occurs if a second tap occurs beofre a time window Twindow expires but not before a hold time Thold after the occurance of the first tap. In the case of Accelerometer, an on chip counter is available. Typically the counter would be running at 32KHz clock. This counter value is used to track the timing intervals required for this application. To simulate IP Core of PSK ,Test Bench is developed. In Test Bench required clock, stimulus and control bits are generated to simulate the IP Core. Figure 7.1 shows Test Bench simulation result of IP core as BPSK Modulaator on Modelsim Simulator.

5.1 Pseudo Code

wait_for_measurement

read_from_reg (Ax,Ay,Az)

if mod(Ax) or mod(Ay) or mod(Az) ¿ threshold

read_from_reg(T1) //read from counter regeister
goto Case1

Case1 : single tap detect
read_from_reg(T2)
if (T2-T1 ¿ t_tap_int)
break
else if (mod(Ax) ¡ threshold and mod(Ay) ¡ threshold and mod(Az) ¡ threshold)
single_tap_detect = 1
goto Case2

Case 2: double tap detect
read_from_reg(T3) if(T3-T2 ¿ t_window)
break
else if (mod(Ax) or mod(Ay) or mod(Az) ¿ threshold
read_from_reg(T4)
if (T4-T2) ¿ t_hold
goto Case3
else brak
Case 3: double tap detect
read_from_reg(T5)
if(T5-T4 ¿ t_tap_int)
break
else if (mod(Ax) ¡ threshold and mod(Ay) ¡ threshold and mod(Az) ¡ threshold)
double_tap_detect =1
single_tap_detect =0 end

5.2 Execute Path

//ASUMPTIONS //1) MTIM16 works @ 32KHz. //Doubt //1) Where to program the frame time?

//Basic Info //If the device is enabled for tap/double //tap detect application, sensed value //is used to determine the tap occurrence. //Tap is said to have occurred if any of //the sensor output crosses a predefined //threshold and returns back to the normal //value within a predefined time interval T_TAP_INTERVAL #define AFE_BASE_ADDR 24'hFF_E040

#define PDB_BASE_ADDR 24'hFF_E000

#define AFE_CSR0 AFE_BASE_ADDR+8'h00

#define AFE_CSR1 AFE_BASE_ADDR+8'h01

#define AFE_XACC0 AFE_BASE_ADDR+8'h02

#define AFE_XACC1 AFE_BASE_ADDR+8'h03

#define AFE_YACC0 AFE_BASE_ADDR+8'h04

#define AFE_YACC1 AFE_BASE_ADDR+8'h05

#define AFE_ZACC0 AFE_BASE_ADDR+8'h06

#define AFE_ZACC1 AFE_BASE_ADDR+8'h07

#define FS(15:14)

#define C4S(13:12)

#define CM(11:10)

#define AAF(9:8)

#define ST 7

#define SWTRIG 1

#define COCO 2 //replace PDB with MTIM16

#define PDB_CSR PDB_BASE_ADDR+8'h00

#define PDB_DELAYA PDB_BASE_ADDR+8'h01

#define PDB_DELAYB PDB_BASE_ADDR+8'h02

#define PDB_COUNT PDB_BASE_ADDR+8'h04

```
    #define PRESCALER [15:13]
#define SB 12
#define SA 11
#define IENB 10
#define IENA 9
#define BOS [8:7]
#define AOS [6:5]
#define CONT 4
#define SWTRIG 3
#define TRIGSEL [2:1]
#define EN 0

    #define T_TAP_INTERVAL 16'h00;
#define T_HOLD 16'h00;
#define T_WINDOW 16'h00;

    function tap_detect(void)
int high_count;
int X_NEW;
int Y_NEW;
int Z_NEW;
int single_tap_detect;
int tap; //initial setting in the ADC
//by setting C4S=00 in AFE_CSR,
//select the acceleration output as input to ADC
write_reg(AFE_CSR_C4S,"0x00");

    //Enable the start_phiD interrupt which will
//bring the CPU out of the reset
```

write_reg(FCSR_SFDIE,'1');

//After the initial setup is establish for AFE
//go into stop mode by executing STOP instruction
write_reg(STOPCR_FC,'1');//select STOPfc in SIM as it is phi_A phase. execute_stop;
//now the CPU will come out of interrupt //on encountering start_phiD.

//read values of output of X, Y, Z transducer after conversion // Declarations of variables???
read_new_value();

//check if accerelaration detected is above the threshold or not // threshold should be a "#define" or picked from some place in memory if((mod(X_NEW) or mod(Y_NEW) or mod(Z_NEW))¿threshold) //start counter using software trigger
// Check if MTIM needs to be used instead. Also confirm the usage of MTIM.
// To ensure that you get both the interval within "TAP_INTERVAL" and the
// interval outside it. write_reg PDB_CSR[EN] = 1;//enable pdb
write_reg PDB_DELAYA = T_TAP_INTERVAL;//set the single tap duration
write_reg PDB_CSR[TRIGSEL] = "00";//select pdb trigger source as software trigger
write_reg PDB_CSR[SWTRIG] = 1;//give software trigger
//go into stop mode by executing STOP instruction write_reg(STOPCR_SC,'1');//select STOPsc in SIM as it is phi_I phase execute_stop;

while(reg_read(PDB_COUNT)¡=T_TAP_INTERVAL) read_new_value();
if((mod(X_NEW) and mod(Y_NEW) and mod(Z_NEW))¡ threshold)
tap=0;
break;
else
tap=1;

write_reg(STOPCR_SC,'1');
//select STOPsc in SIM as it is phi_I phase execute_stop;

write_reg(STOPCR_SC,'1');
//select STOPsc in SIM as it is phi_I phase execute_stop;

read_new_value();

if((mod(X_NEW) and mod(Y_NEW) and mod(Z_NEW))¡ threshold) single_tap_detect=tap;
else single_tap_detect=0;

write_reg(STOPCR_SC,'1');
//select STOPsc in SIM as it is phi_I phase execute_stop;

function read_new_value()
while(!reg_read(AFE_CSR_COCO));
X_NEW=reg_read(AFE_XACC);
Y_NEW=reg_read(AFE_YACC);
Z_NEW=reg_read(AFE_ZACC);

Chapter 6

Flip detection At System Level

If this application is enabled, the application software uses the sensed values to determine if the object is flipped to either of the six possible directions i.e. +X,-X, +Y,-Y, +Z,-Z The flip detection algorithm described here detects if the flip has occurred or not and it also determines the primary direction of the flip. The flip occurrence is said to have happened when the acceleration along any of the six directions crosses a predefined threshold value. Once the threshold is crossed, next task is to determine the amplitude of the acceleration and the direction that records the highest amplitude is considered the primary direction for flip

Pseudo Code

wait_for_measurement
Axfst=Axsnd=Ayfst=Aysnd=Azfst=Azsnd 0
//initialization
read_from_reg (Ax,Ay,Az)
if mod (Ax) or mod (Ay) or mod (Az) ¿ threshold
goto Case1
Case 1:
if (Ax ¿0 and Ax ¿ Axfst)
then Axfst = Axsnd= Ax, dir = posx,

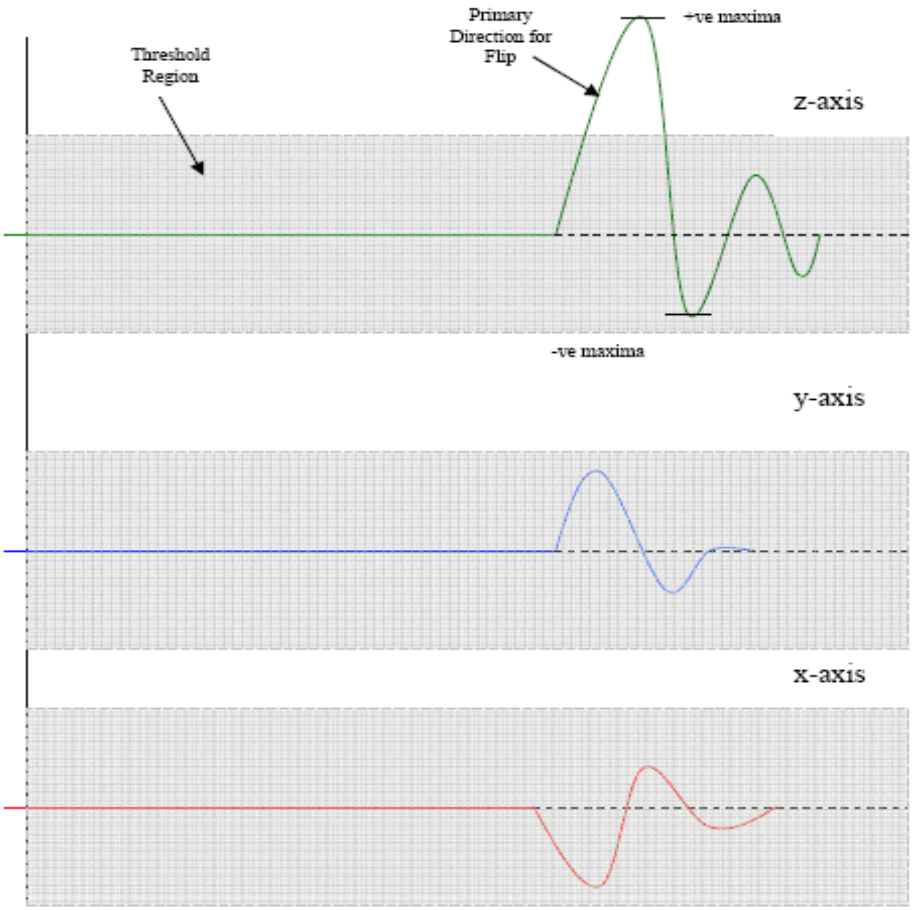

Figure 6.1: flip Detection

goto Case1

if (Ay ¿0 and Ay ¿ Afst)

then Ayfst = Aysnd= Ay, dir = posy,

goto Case1

if (Az ¿0 and Az ¿ Afst)

then Azfst = Azsnd=Az, dir = posz, goto Case1

if (Ax ¡0 and Ax ¡ Axfst)

then Axfst =Axsnd= Ax, dir = negx,goto Case1

if (Ay ¡0 and Ay¡ Ayfst) then Ayfst = Aysnd=Ay, dir = negy,goto Case1

if (Az ¡0 and Az¡ Azfst)

then Azfst =Azsnd= Az, dir = negz,goto Case1

goto Case2

Case 2:

if (Axfst ¿ 0 and Ax ¡ Axsnd)

then Axsnd = Ax, goto Case2

if (Ayfst ¿ 0 and Ay ¡ Aysnd) then Aysnd = Ay, goto Case2

if (Azfst ¿ 0 and Az ¡ Azsnd)

then Azsnd = Az, goto Case2

if (Axfst ¡ 0 and Ax ¿ Axsnd) then Axsnd = Ax, goto Case2

if (Ayfst ¡ 0 and Ay ¿ Aysnd) then Aysnd = Ay, goto Case2

if (Azfst ¡ 0 and Az ¿ Azsnd) then Azsnd = Az, goto Case2

find max (mod(Axfst Axsnd), mod(Ayfst Aysnd), mod(Azfst Azsnd))

Chapter 7

A Typical SOC Device

Soc has the following major parts.

- CPU
- System Bus
- Memory
- IO modules
- master devices
- Control modules

CPU that run the embedded software. In many cases this CPU is third party CPU, of the king of ARM or MIPS. The CPU is used to run general purpose tasks, such as initialization and configuration and usually high level network protocols. System bus which connects ll sub-modules of the SOC. A device may have more than a single BUS, though we may simplify it using a single BUS model// Memory used for data storage, (the CPU related memory is not sown here and assumed a part of the CPU)

IO modules which connect the SOC device with the external world, such as Ethernet 10/100/1G/10G, USB.

master devices that can initiate a transaction of the system bus, such as a DMA, other processing units or a bridge connected to an external bus. Master devices are like IO module with two exceptions: o They can initiate transfers on the bus o They may issue transaction between sub-modules, not only from/to external pin/buses// Control modules, such as interrupt controllers, clock generation module, etc, do not directly involved in data transfers, but have configuration and status registers that can be read and written.

When trying to identify the type of data transition within the type of such SOC we may say that basically, there are only two major kinds of data transactions in SOC system.

- TX transaction: from the CPU to the external world/pins, through the IO modules
- RX transaction: from the external pins, through the IO modules to the CPU

In, both kinds of transactions, the data have an "intermediate parking" in registers or memory. Some data transactions, starts or/and ends in registers or memory, like DMA transfers or configurations that are basically write transaction from CPU to memory or registers.From verification point of you, all data transactions are the same and hence have similar implementation.

Each path of data transfer needs:A driver that is connected to the transaction start point (initiator). A collector that is connected to the transaction end point. A checker that automatically compares the collected data to the expected result. The expected result is typically a simple transformation of the input data. A configuration code that initiates the registers related to this path

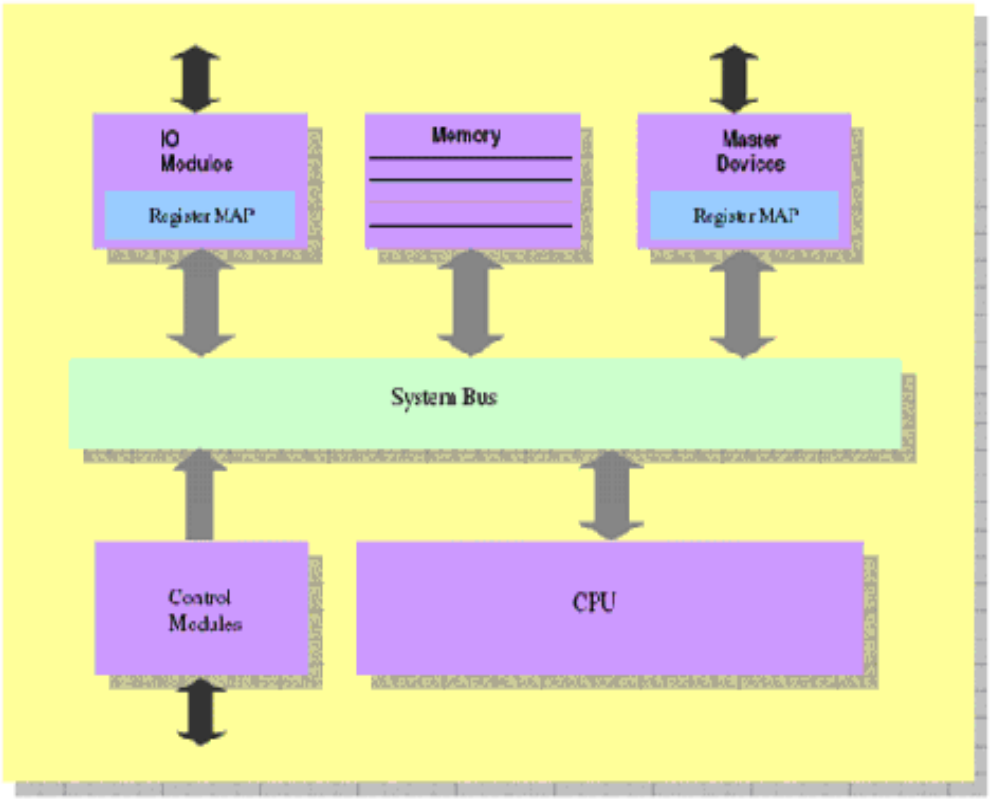

Figure 1: SOC major blocks

Figure 7.1: SOC Device

Chapter 8

SoC Verification Environment

This section contians Verification challenges And Solution , Traditional SOC Verification and Verification Planning Guidelines.

8.1 Verification challenges And Solution

The typical System-On-Chip (SOC) may contain the following components. The processor (ARM or DSP), the processor bus, many peripherals like USB and UART, peripheral bus, the bridge which connects the buses and a Controller. The verification of SOC is a challenging one because of the following reasons.

Integration of various modules : The main focus on verification of SOC is to check the integration between the various modules. The SOC verification engineers assumes that each module was independently verified by the module level verification engineers.

IP block re-use : IP reuse was indeed seen as a way to foster development productivity and output that would eventually offset the design productivity gap. Many companies treat their IPs as an asset.

HW/SW co-verification : An SOC is really ready to ship when the complete application works, not just when hardware simulations pass in regressions. In other words, the ultimate test for a chip is to see it performs its application correctly and completely. That means execute the software together with the RTL. So we need a way to capture both HW and SW activities in the tests we write to verify the SOC.

Some of the SOC bugs might hide in the following areas.

- Interactions b/w the various blocks.
- Unexpected SW/HW handling

All the challenges above indicates that we need a rigorous verification of each of the SOC components separately. I'll explain the trends in traditional SOC verification methodology in the next post.

8.2 Traditional SOC Verification

- Write a detailed test plan document We usually write hundreds of directed tests to verify the specific scenarios and all sort of activities a verification engineer can think it is an important. But there are some limitations
- The complexity of SOC is such that, many important specific scenarios in which the bug might hide are never thought of.
- As the complexity of SOC increases, it become difficult to write a directed test that reach the goals

Test Generations

Each directed tests that we write, check the specific scenarios only once. But this is not advisable. Since we need to exercise these specific scenarios with different combination of inputs, then only we can find the hiding bugs. Many of us write a

random test case to find the hiding bugs, but these are exercised only at the end of the verification cycle. Though these tests reach most of the unexpected corners, we will be verifying the same scenarios again and again and still tend to miss a lot of bugs. But what we actually need is to focus on the particular area of interest in the design. So . We need a generic test generators that can easily directed into areas of interest.

Integration

Test bench development for SOC design requires more efforts than the design itself. Many SOC verification test benches doesn't have a means for verifying the correctness of the integration of various modules. Instead the DUT is exercised as a whole unit. The main draw back to this approach is finding the source of the problems by tracing the signals all the way back to where it originated from takes much time. This leads to the need for integration monitors that could identify integration problems at the source.

Every design and verification team needs an answer for the Million dollar question. When are we ready for tape out?

To answer for this question is very tough as the verification quality is very hard to measure. Every one's answer would be different. My answer would be depends on code, branch, expression and toggle coverage, functional coverage and bug rates. To solve this dilemma, there is need for coverage metrics that will measure progress in a more precise way.

To summarize, there is always an element of spray and pray(luck) in verification, we are hoping that we will hit and identify most bugs. In SOCs, where so many independent components are integrated, the uncertainty in results is greater. There are new technologies and methodologies available today that offer a more dependable process, with less praying and less time that needs to be invested. In the next post i'll explain unique approaches in SOC verification

8.3 Verification Planning Guidelines

In the last post, we saw the traditional SOC verification approach. In this post, we are going to see the unique approaches in SOC verification. SOC verification becomes more complex because of many different kinds of IPs on the chip. A good understanding of the overall application of SOC is essential.

The following should be considered in the verification planning. External Interface Emulation When you verify the complex SOCs, you should consider the full chip emulation. The external interface of each and every IPs on the SOC as well as the SOC data interfaces should also be examined. This should be performed simultaneously for all cores.

Unit level to Top level

SOC designs are built from bottom to top. The truth is that the unit level must be used in any of the design hierarchy imposes a need to verify these modules in any possible scenarios.

Re-Use the verification components

As the leaf modules are assembled to create the SOC, many of the leaf module interfaces are internal interfaces between various modules of SOC, and there is no longer need to drive their inputs. However other interfaces are external interfaces to SOC. If the test generators for external interfaces are independent components, then most system level stimuli can be taken as is from the various module environment.

Many components in SOC can work independently and work in parallel with other components. In order to exercise the SOC in corner cases, the tests should be able to describe parallel streams of activity for each component separately.

Integration Monitors The primary focus of SOC verification is on integration. Most bugs appear in the integration b/w blocks. An integration monitor that comes with an IP can be great help to find the integration problem. It can be hooked in to the simulation environment and just run to see any integration violation appears on the

monitor. This can save the time dramatically. This kind of IP monitors can bring lot of benefits in quality of SOC.

Coverage It is important tool for identifying areas that were never exercised. Both code and toggle coverage are the first indication for areas that were never exercised. However they never tell you that you achieved the full verification. Functional coverage allows you to define what functionality of the device should be monitored.
Looking at functional coverage reports, you may conclude that certain features were already exercised and focus your efforts on the areas that were neglected. But most significant impact of functional coverage in context of SOC verification is in eliminating the need to write many of the most time consuming and hard to write tests.

We discuss two specific classes of test-cases. These are test-cases for verifying the memory modules and the test-cases for verifying the data transfer modules. These are considered since they form a significantly large subset of the device functionality. We implement a prototype test-case generator and also present an example to illustrate the use of methodology for each of these classes. The use of our methodology enables (i) the creation of test-cases automatically that are correct by construction and (ii) re-use of the test-case code segments from one SoC to another. Some of the properties (of the modules and the SoC) presented in our work can be easily made part of the architectural specification, and hence, can further reduce the effort needed to create them.

8.4 SoC Verification Flow

A typical SoC verification flow consists of three major tasks; modify, test and evaluate. The diagram below depicts the flow and the various processes carried out during SoC verification. Designers follow this iterative loop of modification, testing and eval-

uation until the verification objectives are met.

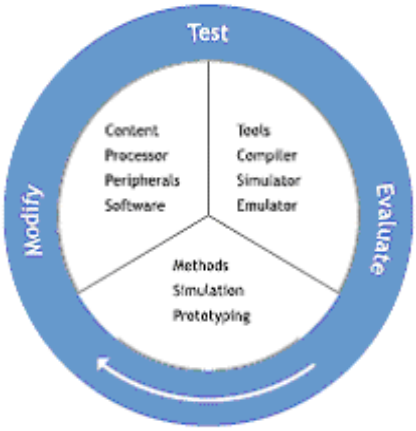

Figure 8.1: SoC verification flow

In the test phase, verification methodology and tools have major impact on reduction of the Iterative Loop circumference. we will discuss a typical SoC design that contains an processor, memory, and custom RTL logic. The processor communicates with memory and peripherals using an bus interface.
The processor model is a design-signoff model (DSM) and all the peripherals and additional blocks in the design are included as RTL models (cores). Some of the RTL blocks / peripherals act as a secondary bus master to the processor.
HW/SW Co-Verification and Test Generation.

A traditional verification and test approach allows software verification only once the silicon is out from the foundry. This makes hardware and software debugging tasks sequential, increasing the product development time.
Verification is to make the hardware and software debugging tasks as concurrent as possible. However, once the teams discover the benefits of co-verification they artificially believe that the design is bug free if the design runs all of the diagnostics, boots the operating system, and runs applications. Unfortunately, software changes

frequently and there is little guarantee that the software exercises complete features and functions of hardware. A better approach to verification is by running a combination of software tests to verify the complete bus interface. Using the above example and co-verification techniques this can be accomplished in two steps. First, it is possible to run software programs on an co-verification model and capture the resulting sequences of transfers created by these software programs .
Secondly, the transactions written in the form of a command file are used to drive a synthesizable bus functional model of the protocol. These commands file contains AMBA transaction commands, sideband commans, and delay commands. The transfer descriptions in this file have all the information needed to reproduce the sequence of operations as seen by the hardware design without requiring a full processor model. In the above application, co-verification is used to provide stimulus generation for a bus functional model.

Random Test Case Generation

To achieve full functional coverage its imperative to generate high speed stimulus. For high speed, the BFM is recommended again. Transactions encapsulate the necessary address, data, and control information. On the next page shows transactions generated using a C program. With this architecture, tests can be directed, random, or a blend of both. The transaction files produced by software tests can be combined with random test generation to construct a comprehensive stress test for the hardware design. The solution offers benefits from the synthesizable bus functional models that can also be used for simulation acceleration and emulation. The subsystem is combined with other synthesizable models and in-circuit interfaces to create a complete verification environment. The C-API of the synthesizable model also provides the necessary measurements for coverage. The verification team can use these metrics to adjust the random generation constraints and rerun the simulation to check the coverage improvement.

Coverage Analysis

To analyze the coverage of an transaction model, an coverage model, constraints applied to the transaction model, and a reference model are introduced.

Reference Data for Checking

Self checking tests are the best way to automate verification. The simplest way to do this is to use system data to check the validity of the design. To accomplish this, a reference model for the design is used. When a read occurs, the data returned from the reference model is sent to the BFM as the expected data during the read. The comparison is done in the BFM to maximize performance in simulation acceleration and emulation applications. A single bit reflecting the comparison is returned to the test program. The reference models can be implemented in C or Verilog for expected results.

Software Advantage

Using software as part of the verification, ensures the design will act in the same way as the final product. This stimulus will be "realistic". That is it will put the design being tested through its typical operations. This enables support of system level co-verification of tests for the complete collection of components in hardware, software or combination of the two, uncovering problems that would never be found in isolation. isolation. The key in achieving the performance is to filter out code and data references that are not relevant to the operation of the hardware this is done by implementation of all monitors in Verilog. Also, using C code in Design Verification tests can be reused on the lab bench. Using software as a stimulus is faster and easier to create than writing stimulus in HDL.

Test case Development

Test case development involves writing test cases in C along with Verilog test bench. In order to allow synchronization between the C test code run by the processor and

the Verilog test bench, a "Porthole mechanism" is supported in the simulation environment. Porthole mechanism allows using dedicated reserved addresses in the memory map. When writing to those addresses from the C test, the verilog environment will detect the address bus change and accordingly to the value of address and data written, will display the pre-defined message and activate the predefined verilog event. Monitors are used to snoop the processor's internal operations including register status that are implemented in verilog.

IP Stand Alone Verification

Verification is also done bottom-up, as many IP blocks are developed and verified in stand alone while the SoC is defined. Many IP blocks are re-used from pre vious designs, typically up 80 to 90content come from existing blocks requiring minor adaptations and some fine tuning modifications based on experience from existing systems. At the IP level, the verification focuses on stand-alone IP functionality, i.e. building test benches based on application stimulus to ensure that the IP is fully compliant with the specifications. Most of the verification work is performed at the RTL level, while the C++ based description, SystemC or C++ with custom library, are used to develop reference models. The design automation technology offers a comprehensive and efficient set of tools to cover the IP verification needs. Good examples are OpenVera [12], e-language [13], and others.

SoC Verification platform.

At the SoC level, verification focuses on the integration of platforms and IP blocks by verifying signals connectivity, memory map location, connections to the PADs, data path, DMAs, interrupts and inter-process communication either on or off chip (see Figure 2). System level tests will also be developed to verify sub-system behaviors but this is quite limited due to low simulation speed at the chip top level.

Debugging the testcases

The task of the functional verification is to check that the required functionality is implemented by the RTL. Verification of the design is generally done using the various testcases written in HVL(High level Verification Language) due to various advantages of the HVLs.
In the initial phase of the verification a large number of the bugs are found in the RTL as RTL may not be stable. As verification goes further the number of the bugs starts reducing which is desired as the bugs found in the latter stages are very costly.

How ever when any bug is found it is very important to check is that if it is a RTL bug or testcase one. The perception of the specification may differ from designer to verification engineer. This may cause that bug for verification engineer may not be same for designer. So whenever any bug is found it is very necessary to check that if it is due to something done wrong in the testcase.

Understand the specification
As mentioned above that understanding of the specification may be different for verification engineer and the designer. So it is very important that first clear the understanding specification by going through the specification doc and try to understand the expected behavior of design. This requires the patient reading of the specification doc and separate out the important points.
Generate the testcase
Once the specification of design it is next step to generate the testcase. While reading the testcase it is required to understand that which scenario is implemented in it. It is also required to see that all the necessary condition are satisfied by it.
Go through the log file
When testcase is run the log file is generated along with the dump. This log file has various details such as which stimulus are given, which phase of the testcase is running at correspond time, all the info messages printed for debugging purpose, information regarding the coverage and also the various error.

Locate the problem Once the gone through the log file one can list down the errors in the separate file. Then the real task of the debugging starts. One need to locate the error first in the particular section of the testcase. The info messages printed may help in it. Once the erroneous section of the testcase is found then it is required to understand the stimulus given. From this stimulus one should write down expected response of the duv from the specification. Then this response can be compared with the actual one from the dump. From this comparison one can find the mismatch then it is required to debug this mismatch, which leads to the bug. One may need to go up to RTL level so it can be seen that if bug id not from RTL

Make the corrective action.

Once the bug find, if the bug is from testcase then proper correction should be made to it. If the bug is from the RTL then it has to be reported to the RTL team and it is fixed.

Re-verify testcase.

Once the bug is fixed we need to confirm that testcase which failed previous now passes.

Chapter 9

Modulo Timer

This section summarizes the verification efforts for the 16bit Modulo Timer (MTIM16) module.
The primary verification tasks consisted of:

- MTIM16 Block User Guide (BUG) functional description specifiication annotation.
- Verification Testbench and Pattern Development Process.
- HDLScore code coverage generation and analysis.
- RTL Regressions.
- Sea-Of-Gate (SOG) SDF Simulation and DVT Regressions.
- Regression Report.

9.1 Introduction

The 16bit Modulo Timer (MTIM16) module verification is accomplished by using directed functional patterns. A verification environment has been built using interface

monitors and external driver tasks, along with a comprehensive set of self-checking functional patterns. The verification environment is written for the S08SIM simulation environment. The patterns are written in C language with embedded verilog.
Testbench Overview.
This section describes the S08 testbench used to verify the MTIM16 module. In this verfication methodology all modules are verified in a full-chip environment. Figure 4-1 shows the MTIM16 module with testbench behavioral model, monitors, drivers, and tasks. The testbench is configurable on a simulation case by case basis.

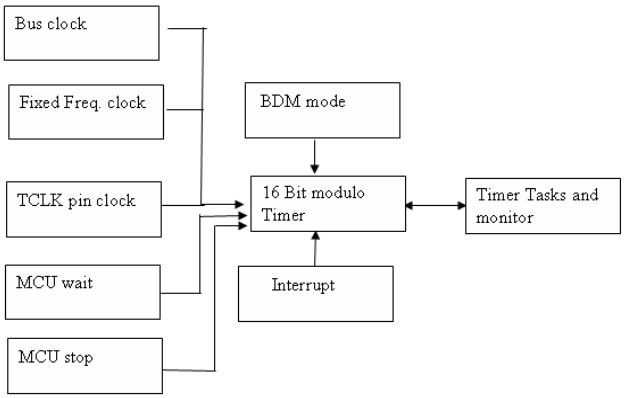

Figure 9.1: Timer Module Connection

The drivers and monitors used specifically for the MTIM16 module simulations are listed in this section. Refer to Global Tasks in the S08sim Users Guide for syntax and descriptions of basic tasks of the S08 simulation environment. In addition, the MCU monitors listed in the S08sim Users Guide were used.

9.2 Stimulus

Stimulus files are named according the following convention: The mode indicator is completely removed if the pattern may run and pass in both User and Test modes. These patterns are written to verify the same functionality in both User and Test modes with the intent that the User mode pattern may be removed from the Production test suite with the functionality still overed. Refer to the s08sim Users Guide Types and Format for additional information on the format of the stimulus files. covered. The assembly pattern file extension is .asm. The assembly code is synchronized with the Verilog to allow reset to be pulled, signals to be driven or checked, and the simulation to end. The directory structure is consistent with Stingray.

9.2.1 Stimulus Organization

All patterns are written to run at a system level, in Assembly mode on a S08 Core, and with internal memory since no external address bus is available on some chips. For a list of stimulus types, please see Table 7-1. Patterns are identified with the intent of being run in simulation, on the tester, and/or in production as indicated in the Target column with any (or all) of the following: S = Pattern will be run in simulation regressions (rtl, gate, sog). T = Pattern can be run on tester. P = Pattern must be run in production (i.e. detects fault coverage on a hard block). T* = Pattern will be run on tester but not tester readyTable Stimulus Type Classification Stimulus Type Description Timing Critical .
Stimulus that is used to verify a specific speed path for a given IP Block IP/VC Block Dependent Stimulus that is used to verify operation between IP Blocks Minimum Pin Set Stimulus that only relies on the minimum pin set to be present.

9.2.2 Stimulus Template

if the pattern is to run in any (or multiple) User mode(s) as defined by the integration module b. tst if the pattern is to run in any (or multiple) Test mode(s) as defined by the integration module The mode indicator is completely removed if the pattern may run and pass in both User and Test modes. These patterns are written to verify the same functionality in both User and Test modes with the intent that the User mode pattern may be removed from the Production test suite with the functionality still covered.

9.2.3 Stimulus Directory Structure

The directory structure is consistent with Stingray.2006.10 7.4 SoC Stimulus List Describe and list the tests that can be run at the system level to verify the VC. Specify all stimulus names, the type to which each stimulus belongs, and describe each stimulus. Use the table format from Table.

9.3 Coding for Verification

The goal of this section is to describe the standards for VC and system-level testbenches. The following rules and guidelines define a testbench architecture that consists of standardized interfaces between components. Figure shows the recommended testbench architecture.

Monitor - Observes and checks DUV interface protocol and abstracts signal transitions into events that are published for other testbench components.
Driver - Drives transactions onto the signal interface based on commands received from the transactor.

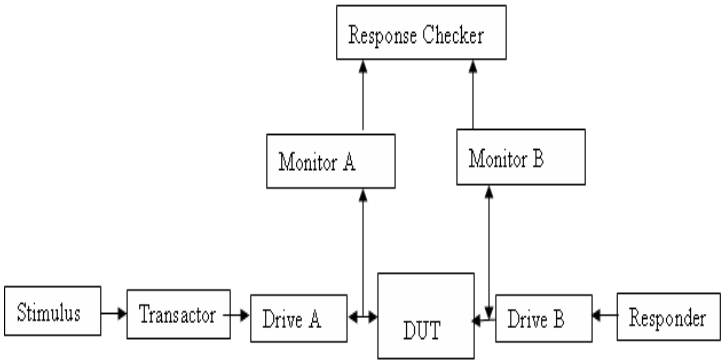

Figure 9.2: Testbench Architecture

Transactors - Translates and coordinates sending commands to drivers from stimulus. Stimulus - Creates and issues commands to transactors. Responder - Subscribes to events from monitors and uses drivers to initiate appropriate transactions in response to events Coverage - Collect coverage metrics " Response Checker - Checks DUV behavior.

Drivers

This section defines standard coding practice for VC drivers used in functional verification. Drivers may respond and drive the interface of the VC by accepting commands in the stimulus or events from monitors. The drivers may also be stand-alone elements. Stand-alone drivers are reusable without dependencies on the testbench and stimulus control.

Responders

This section will define the standard coding practice for VC responders used in functional verification. Memory array models are verification components that represent the functionality of internal and external memory circuits. Behavioral models are used instead of implementation-oriented models to speed up sim-ulation. The models

are reusable VC. Reusability is enhanced by following the rules and guidelines in this section.

Response Checkers

This section will define the Response Checker, a testbench component that ensures the requests coming into a VC are responded to correctly. Block-level response checkers verify that operations coming out of a VC should be happening and that the results are correct. The response checker must be configured independent of the VC Block-level response checkers must be configured via the testbench, stimulus, and/or monitors. Configuration of the checker must not occur based on the internal VC state. Checkers may be used to preload states into the VC. Reason: Problems with the VC configuration may be masked. This provides independent verification of the VC operation and configuration mechanism. Deliverables: V4 Properties: (NewIP=='True') Response checkers should not connect to the VC Response checkers should interface with existing testbench components such as monitors. Reason: This makes the response checker more maintainable. If the interface changes, only the monitor is required to change, and the interaction with the response checker may be preserved as-is. Deliverables: V4 Properties: (NewIP=='True') Response checkers should publish coverage events Reason: Response checkers often contain complex response calculation code that can shared to report interesting coverage events.

This section outlines rules that apply to stimulus regardless of stimulus form. Stimulus could be slave mode or master mode, random or directed, depending on the context. Random simulation is used for ex-ercising boundary cases of the VC. It is achieved by generating pseudo-random transactions for stimulus. Randomness can be either in content (e.g., random data in a write transaction), appearance (e.g., randomly choose between a read and a write transaction), or both. Constraints are written to specify the range of allowed random transactions. Tools are used to randomly generate transaction within the allowed range. Probability and weighting schemes are used to bias the ran-dom selection of transactions. Properly coded stimulus can

produce stimulus that are portable and easier to maintain. The following stan-dards and guidelines are for verification source code. Partitioning can impact the ease with which a model can be adapted to an application. Patterns must be partitioned to facilitate portability to different chips. Proper partitioning allows the easy omission of stim-ulus whose functions and/or pins are not being utilized on a particular chip. Patterns are able to be used as is without modifications. This directly reduces stimulus debug time.

Simulation Environment.

This section lists requirements for the regression environment to enable verification reuse. Regression standards are essential to ensure that stimulus can be run in an automated manner. This section provides standards and guidelines on running simulations using VC that is checked out of the IP Repository. A stan-dard represents a practice that must be supported by the simulation environment. Regression guidelines are the most desirable approach to particular issues with running simulations, but are not mandatory for the simulation environment to support them. Efficiently running regressions is an important verification task. Regression scripts may be written to au-tomate the task of running regressions and gathering results. Regressions must be allowed to be batched off to several machines on the network to take advantage of machine resources. In addition, the ability to quickly analyze the results will enhance the productivity of the verification person. Regressions may be run at the VC or SoC level. It is essential that the regression environment support both levels of regression testing. The regression environment must also provide the flexibility to run regressions using various testbench configurations, various VC or SoC views, running all types of stimulus, and com-pare current simulation results against reference results.

Code and Functional Coverage.

General Coverage is used as a metric to judge the quality of the verification. It attempts to show which behaviors have been simulated and which have not. Code

coverage is used to measure how much of the code is ex-ercised. Functional coverage is used to measure how many of the design features have been exercised. No commonly used coverage metric is perfect. This means that a report of 100

Assertion-Based Verification

An assertion is a mathematically precise property written in a machine-readable formal language. Exam-ples of formal assertion languages are Freescale CBV, Synopsys OVA, Accellera SVA and PSL, and cer-tain subsets of Verilog for which formal semantics have been given. Assertion-based verification uses assertions to define properties, such as assumptions and obligations, of a design. The assertions themselves are formal properties, but the verification computations can be formal (e.g., model checking), semi-formal (e.g., bounded model checking), or simulation-based (e.g., assertion monitoring in testbench-driven or constraint-driven simulation). Formal computations have the potential to provide a mathematical proof that an assertion holds.

Chapter 10

Programmable Delay Block

Features

Positive transition of a trigger input will initiate the counter. The trigger source is software programmable to be one of :

- phi A started
- phi B started
- software trigger

Supports two output signals. Each has an independently controlled delay from the trigger input.Digital comparator outputs can be can be used to schedule precise edge placement for pulsed output. Continuous trigger or single shot mode supported .Each output is independently enabled.

Modes of Operation

Modes of operation include:

- Disabled: Counter is off and both A and B outputs are low.

- Enabled OneShot: Counter is enabled and restarted at count zero upon receiving a positive edge on the input trigger. A and B will see only one output transition per input trigger.
- Enabled Continuous: Counter is enabled and restarted at count zero. The counter will be rolled over to zero again when the count reaches the value specified in the MOD register, and counting restarted. This enables a continuous stream of output pulses as a result of a single trigger input

Block Diagram

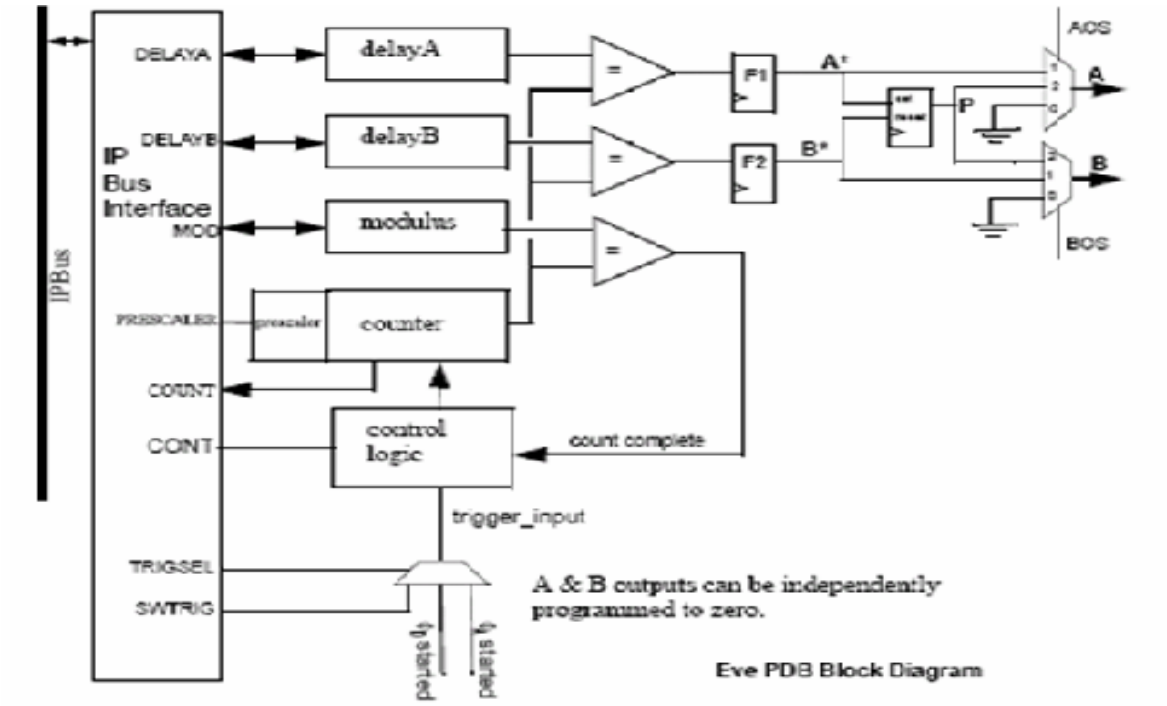

Figure 10.1: Block Diagram

The pulsed mode is shown in Figure In this case, A* and B* are used to precisely schedule the rising and falling edges for the output waveform.

- Registers Descriptions

PDB Control and Status Register (CSR)

This register contains status and control bits for the Programmable Delay Block. The counter is enabled if EN has been set to one.

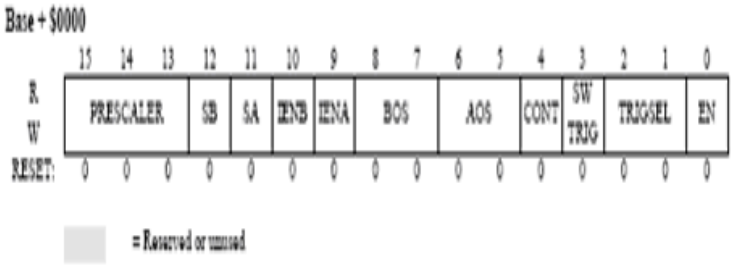

Figure 10.2: Programmable Delay Block Control and Status Register (CSR)

- PDB Delay A and Delay B Registers (DELAYA and DELAYB)

These registers are used to specify the delay from assertion of TriggerIn to assertion of A and B out. The delay is in terms of peripheral clock cycles.

- PDB Modulus Register (MOD)

This register specifies the period of the counter in terms of peripheral bus cycles. When the counter reaches this value, it will be reset back to all zeros. If CSR [CONT] is set to one, the count will begin anew.

- PDB COUNT Register (COUNT)

This register can be used to read the current value of the counter. It is READ ONLY.

Additional trigger events, after the first, but before the timer times out, will cause the counter to restart. Impacts of using the prescaler on timing resolution Use

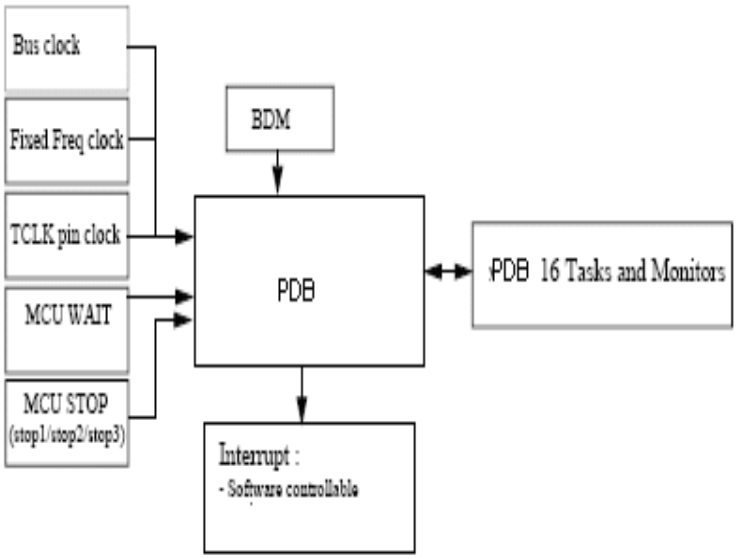

Figure 10.3: PDB connection

of prescalers ¿ 1 limit the count/delay accuracy in terms of peripheral clocks (to the modulus of the prescaler value). If the the prescaler is set to div 2 then the only values of total peripheral clocks that can be detected are even values, if div is set to 4 then the only values of total peripheral clocks that can be decoded as detected are mod(4) and so forth. If the user wanted to set a really long delay value and used div 128 then he would be limited to an resolution of 128 bus clocks. Therefore use the lowest possible prescaler for a given application.

Chapter 11

Summary

- I have implemented the program for the software based application and done the analysis for memory size, total number of cycle and average current. From that analysis CPU programmer can decide which type of the application CPU can run and when it's requirement. In some application I have design without coldfire processor and with coldfire processor, then compared the memory requirement for running the application
- This analysis also give the idea for the partitioning of the Software and Hardware for the any particular application.
- These applications give to the user better facility and protection to it's mobile.
- These applications are also use in Personal Navation Devices , Pedometry and Gaming and Toys
- Most of the modules are reuse from the previous chip.From the SoC level verification , Designer can know the changes of the module's design as per the application requirement. Here some application require delay ,timer , interrupt. So I have done the SoC level verification of modulo timer , programmable delay bolck and interrupt controller. From this verification I gave the suggestion to designer to change the module design as per application requirement.

Application	*Memory (bytes)*	*Cycles With Coldfire*	*Cycles Without Coldfire*
Acceleration Detection	*146*	*65*	*36*
Freefall Detection	*298*	*170*	*82*
Flip Detection	*523*	*211*	*97*
Tap Detection	*671*	*234*	*109*
Double_Tap_Detection	*152*	*76*	*43*
Turnover Detection	*627*	*226*	*136*
Rolling Dice	*1553*	*560*	*385*
slop Detection	*790*	*240*	*167*
Swing Detection	*1617*	*519*	*363*

Table I: Comparison Table

References

References

Index

bit splitter circuit, 10