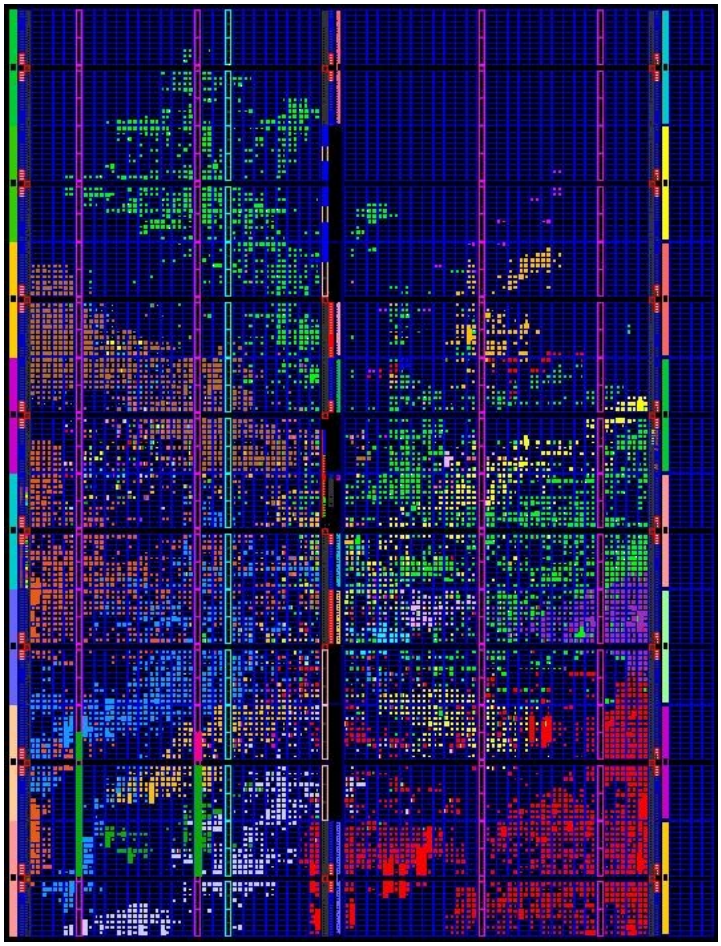
CHALMERS



FPGA prototyping of the MSP430F5172

Master of Science Thesis in Integrated Electronic System Design

Charuchandra Prabhushankar

at Texas Instruments, Deutschland

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, August 2010

FPGA prototyping of the msp430f5172

Charuchandra Prabhushankar

Examiner: Prof. Lars Bengtsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: Colour map of different hierarchical modules of the MSP430F5172 after PAR]

Department of Computer Science and Engineering
Göteborg, Sweden August 2010

1 Abstract

The aim of this master thesis is to develop a working FPGA prototype of the MSP430F5172. This report deals with the standard steps of FPGA recoding. It relates the implementation results to the FPGA logic primitives and an introduction to such logic primitives is provided when such a discussion is presented. So the result and the theory are interconnected throughout the content of the report. It also presents a discussion on the extensive spectrum of tools involved in the FPGA implementation process from EDIF netlist generation to the final bitstream generation as well as analysis tools like timing analyzer. An overview of the principles involved in the implementation process provides a better understanding of the operation of the tools. In addition many features of the tools explored during the execution of the thesis is documented that enables the reader to reduce the learning curve involved in implementing such a project in the future.

The device utilisation for the MSP430F5172 prototype on the Virtex5 LX110 was about 25%. In spite of such a moderate device utilisation the runtimes were as high as 3 hours which can give an idea of the complexities involved in the RTL code of modern microprocessors. Issues faced in the course of the thesis are also presented and discussed. Procedures and principles that can help overcome this issues are also presented.

Finally, the FPGA prototype is subjected to basic functional tests. The results and code example is also provided.

Contents

List of Figures

List of Tables

List of Abbreviations

1. ADC - Analog to Digital Converter
2. ACLK - Auxiliary Clock
3. BOR - BrownOut Reset
4. BRAM - Block RAM
5. BUFG - Global Buffer
6. BUFR - Regional Buffer
7. CLB - Configurable Logic Blocks
8. CMT - Clock Management Tile
9. CPLD - Complex Programmable Logic Device
10. CPU - Central Processing Unit
11. DAC - Digital to Analog Converter
12. DCM -Digital Clock Manager
13. DCO - Digital Controlled Oscillator
14. DMA - Direct Memory Access
15. DRAM - Dynamic RAM
16. DRC - Design Rule Check
17. DUT - Design Under Test
18. EDIF - Electronic Design Interchange Format
19. FF - Flip Flop
20. FPGA - Field Programmable Gate Array
21. GUI - Graphical User Interface

22. I2C - Inter IC
23. ILA - Integrated Logic Analyzer
24. IO - Input Output
25. IOB - IO Block
26. LPM - Low Power Mode
27. LUT - Look Up Table
28. MCLK - Master Clock
29. MUX - Multiplexer
30. NRE - Non Recurring Engineering
31. PAR - Place And Route
32. PC - Program Counter
33. PLL - Phase Locked Loop
34. POR - Power On Reset
35. PUC - Power Up Clear
36. QOR - Quality Of Result
37. RAM - Random Access Memory
38. RISC - Reduced Instruction Set Architecture
39. RTL - Register Transfer Level
40. SMCLK - Sub Master Clock
41. SPI - Serial Peripheral Interface
42. SRAM - Static RAM
43. SSRAM - Synchronous SRAM
44. STA - Static Timing Analysis
45. SVS -Supply Voltage Supervisor
46. UART - Universal Asynchronous Receiver Transmitter
47. UCS - Unified Clock System
48. VHDL - VHSIC Hardware Description Language
49. WDT - Watch Dog Timer
50. XST - Xilinx Synthesis Tool

2 Introduction

Semiconductor solution for implementing digital logic can be broadly considered in terms of ASIC and FPGA. ASIC which stands for Application Specific Integrated Circuit is designed to serve a custom built solution. On the other hand FPGA - Field Programmable Gate Array are reprogrammable and can be used to implement a variety of functions. The choice for using an ASIC or FPGA depends on the parameters required out of the final product. The conventional comparison between ASIC and FPGA is given in the table 1 as a background for these families.

Table 1: ASIC vs FPGA

Parameter	**ASIC**	**FPGA**
Performance	High	Medium
Power Consumption	Low	High
Capacity	High	Low
NRE	High	Low
Cost	High(in small volumes)	Low
Time to market	High	Low

2.1 FPGA prototyping background

FPGA prototyping is the method of developing a prototype of an ASIC for early view into the silicon behavior and early software development. With time to market determining the profit earned on a product, it becomes more pressing to have an early platform to develop new products. In addition FPGA prototypes perform at nearly the intended device speeds for system verification and bug exploration and fix tests. To get an idea of the loss of product sales as a function of delay time to market consider figure 1 as an approximation[1]. FPGA prototypes provide low cost access with high speed of operation.

On the other hand the disadvantages of FPGA based prototypes are:

- Lack of visibility of internal signals which might hamper the debugging process(compared to simulation).
- Extra effort involved in recoding the ASIC RTL to FPGA RTL.
- The analog modules of the design can only be modeled functionally in the FPGA.

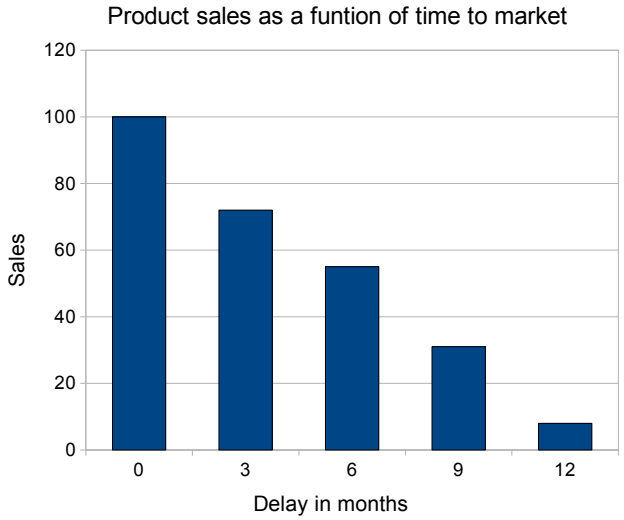


Figure 1: Time To Market (ttm)

FPGA prototyping for initial software development and reducing the probability of silicon failure has become a standard practice. FPGA prototype allows designers and verification engineers to test the device months before the final silicon is available. Software codes for the prototype device can be developed, compiled and downloaded onto the FPGA as shown in an example in the results section 12. Hence FPGA prototyping can be used for a wide variety of crucial functions prior to silicon release.

3 MSP430 introduction

The MSP430 is a 16 bit RISC mixed signal microcontroller designed for ultra low power operation. The MSP430 can be woken up from low power mode and the clocks can be active and stable in less than 1μs which enables the CPU to be active in short bursts of time and spend idle time in low power modes. The MSP430 has a Von-Neumann architecture with a common address bus (MAB) and a common data bus (MDB). Some of the features of the MSP430 are discussed in this section.

Startup sequence: System control module is responsible for reset generation. The MSP430 has three reset signals being generated on startup and these are the brownout reset (BOR), power on reset (POR) and power up clear (PUC). The conditions for the generation of these signals are described here [8]:

A BOR is a device reset. A BOR is only generated by the following events:

- Powering up the device
- A low signal on RST/NMI pin when configured in the reset mode
- A wakeup event from LPMx.5 (LPM3.5 or LPM4.5) modes
- A software BOR event

A POR is always generated when a BOR is generated, but a BOR is not generated by a POR. The following events trigger a POR:

- A BOR signal
- A SVSH (Supply Voltage Supervisor) and/or SVSM low condition when enabled
- A software POR event

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration when watchdog mode only
- Watchdog timer password violation
- A Flash memory password violation
- Power Management Module password violation

Unified Clock System(UCS) and Low Power Mode (LPM) of operation: The salient feature of the MSP430 family of microcontrollers are the low power modes of operation. The low power of operation is possible with the use of low frequency clocks and shutting off parts of the microcontroller in the LPM. The UCS generates three internal clocks for the MSP430 using upto five external sources. The operating modes mainly consider the following trade offs.
Ultra low power— Speed and data throughput— Minimization of individual peripheral current consumption

There are mainly 5 types of low power modes (LPM) as described in the table 2. The UCS module provides the control over the clocking system of the MSP430 to sustain LPM. It includes up to five clock sources:
XT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32kHz crystals or external clock sources in the 4 MHz to 32 MHz range.
VLOCLK: Internal very low power, low frequency oscillator with frequency of about 10kHz.
REFOCLK: Internal low-frequency oscillator with 32kHz typical frequency

DCOCLK: Internal digitally-controlled oscillator (DCO) that can be stabilized by the FLL
XT2CLK: Optional high-frequency oscillator similar to XT1CLK.

Three clock signals are available as outputs from the UCS module:
ACLK: Auxiliary clock. The ACLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK/n is ACLK divided by 1, 2, 4, 8, 16, or 32 and is available externally at a pin. ACLK is usually the source for individual peripheral modules.
MCLK: Master clock. MCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
SMCLK: Subsystem master clock. SMCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. SMCLK can be used to source faster peripherals to frequencies upto 25 MHz [8].

Depending on the LPM different outputs of the UCS is disabled to reduce power consumption. The clocks that are active/inactive in the different modes of operation are presented in the table 2.

Table 2: Low Power Modes

Mode	**Status**
Active	MCLK,SMCLK active, ACLK active
LPM0	MCLK disabled; ACLK active; SMCLK optionally active
LPM1	MCLK disabled; ACLK active; SMCLK optionally active
LPM2	MCLK,SMCLK disabled; dco enabled if it sources ACLK
LPM3	MCLK,SMCLK disabled; dco enabled if it sources ACLK
LPM4	All clocks disabled

Flash memory: The MSP430 flash memory is classified into main memory and info memory. Code or data can be loaded into either of these sections. The info flash is divided into 4 segments A-D. Each segment contains 128bytes. The main memory on the other hand has a segment size of 512 bytes. The logical value of the flash memory after erasing is 1. The bits in the flash memory can be programmed from 1 to 0 but programming from 0 to 1 is not possible and the memory has to be erased for such an operation. The MSP430 flash memory operation include the read and write operation. The flash memory has an integrated controller that controls programming and erase

operations.

Digital IO: The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts. Some devices may include additional port interrupts.
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

The feature of the IO pins can be configured using pxout, pxdir and pxren registers. The overall operation is reflected in the table 3.

Table 3: Port Configuration

PxDIR	**PxREN**	**PxOUT**	**IO configuration**
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

In addition the port mapping functionality allows reconfigurable mapping of port function. The port mapping functionalities include port output of ACLK,MCLK and SMCLK as well as timer a, timerb and USCI functionalities. These port mapping are controlled through registers and can be configured during runtime.

Peripherals: The MSP430 has a wide array of analog and digital integrated peripherals. Many peripherals can function independent of the CPU thus reducing the time the device spends in active mode.

MPY32 Hardware multiplier: The MPY32 supports 8-bit, 16-bit, 24-bit, and 32-bit operands with unsigned multiply, signed multiply, unsigned multiply-accumulate, and signed multiply-accumulate operations. This peripheral does not interfere with the CPU and can be accessed via the DMA.

Direct Memory Access(DMA) controller: It transfers data between memory locations across the entire address range without CPU intervention. This enables better output from the peripherals and more time in LPM.

Watchdog timer: The purpose of a WDT is to perform a system restart in case of software issues. The system reset is generated if the interval of the WDT runs out. The WDT is a 32 bit timer that can be used as a watchdog timer or as an interval timer.

Timer: the timer on the MSP430 is a 16 bit timer/counter. The timer supports multiple capture compares, pwm outputs and interval timings. Timers can also be used as interrupt generators.

Analog to Digital Converter(ADC): It supports 200ksps 12 bit conversion. The ADC is based on successive approximation technique (SAR) and have an internal reference voltage generator of 1.5V or 2.5V. The ADC can sample, convert and store without CPU intervention.

Digital to Analog Converter (DAC): It is a 12 bit voltage output DAC with internal or external reference. It can also be configured for 8 bits of operation.

Communication interfaces supported include UART, SPI and I2C.

4 Hardware used and programming flow

The FPGA board along with the programming interface and the software was provided by Luminary Micro. It has all the components needed to program the FPGA which include:

- Programming interface - in our case programming through the USB interface.
- CPLDs for interfacing with the FPGA .
- Memory to store the FPGA configuration and programming data.
- Clock crystal for FPGA operation and configuration.

- Additional connectors for accessing the FPGA ports.
- Power supply and voltage regulator.

The FPGA hardware system consists of a base board and a FPGA board. The FPGA board has the capability to stack multiple FPGA boards using Samtec connectors. The FPGA board has a Virtex 5 FPGA, the SRAM , clock crystal and LED indicators for the state of the FPGA during configuration. The SRAM used to store the configuration data of the FPGA is of size 8Mbytes and is configured to hold two different FPGA configuration files.

Generally, the base board functionally satisfies the routing of signals using the CPLDs. The CPLDs are used mainly for multiplexing and signal routing on the falcon board. CPLDs are non volatile and hence need to programmed only once. CPLD programming can be achieved by using Verilog RTL which essentially describe the necessary connections. The resulting jed file is used to program the CPLD. The base board also provides communication interfaces such as USB,RS232, ethernet, CAN, JTAG and debugging facilities via chipscope as well as an LCD interface.

Programming: Programming of the FPGA board is through an USB interface to the PC. The interface on the FPGA side is handled by the FTDI: FT2232C which is a dual USB to uart/fifo. The programming of the FPGA is done through a SRAM programmer. The programmer essentially programs the SRAM on the FPGA boards and later after the FPGA is set into master mode where it reads the configuration data from the flash memory.

5 Virtex 5 introduction

For this thesis the FPGA used was the Virtex 5 series which is based on 65 nm technology node[1]. The particulars of the Virtex 5 used is LX110 speed grade -1. However there are other vendors of FPGA which include Altera, Actel, Lattice and Atmel.

Startup process and configuration: Configuration is the process of implementing the intended digital function with the help of a bit file[2] on the FPGA. The bit file is loaded into the internal memory. This memory is volatile and has to be reconfigured

[1]technology node refers to the size of transitors on a chip, more specifically to the width of the channel

[2]which is obtained as a result of synthesis,map, place and route, bit stream generation of the digital system RTL[register transfer level] files

after every power up cycle of the FPGA. The bit file for the MSP430 design is about 3 Mbytes.

There are several modes available to load the configuration data into the FPGA via configuration pins. Mainly there are two modes master and slave. In the master mode the configuration clock (CCLK) is generated by the FPGA but in the slave mode the CCLK is generated by the external data source which provides the bit data. In the serial configuration mode where one configuration bit is loaded every CCLK cycle there are four methods [14].

- Master
- Slave
- Serial Daisy chain
- Ganged serial

The configuration process follows the sequence shown in figure 2 :

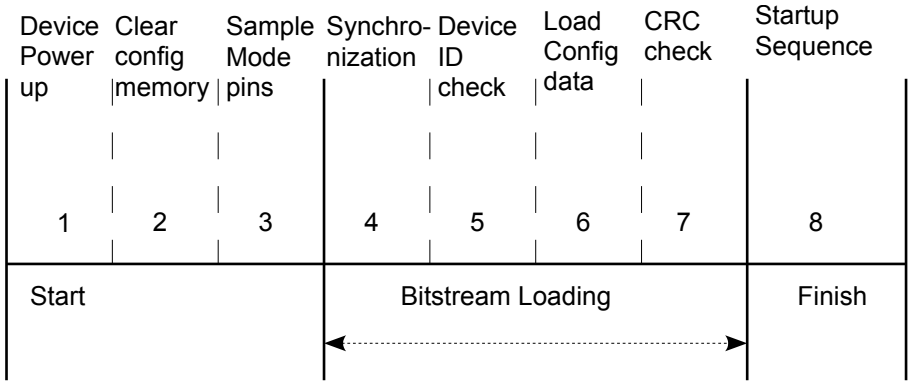


Figure 2: Configuration sequence

1. Detects power-up (Power-On Reset) or PROG being Low.
2. Clears the whole configuration memory.
3. Samples the mode pins to determine the configuration mode. (Master or slave, bit-serial or parallel etc).
4-7. Loads the configuration data starting with a synchronization word and a check for the proper device code and ending with a cyclic redundancy check (CRC) of the complete bitstream.
8. Start-up executes a user-defined sequence of events: releasing the internal reset (or

preset) of flip-flops, optionally waiting for the DCMs to lock, activating the output drivers, releasing DONE pin and asserting end of startup[14].

CLB and slices: The CLBs are the logic building blocks of the FPGA. Each CLB contains two slices. The slices contain four LUTs and four storage elements or flip flops as well as Muxes and carry logic.

Additionally the slices are further divided based on their function into slicel and slicem. Slicem being a superset of slicel has additional function of a distributed ram and 32 bit shift register. This section provides an overview of the Virtex 5 resources.

Table 4: Virtex5 CLB resources

device	CLB array rowxcolumn	number of 6 input LUTs	number of flip-flops
xc5vLX110	160x54	69120	69120

Look Up Table (LUT): LUT are used to implement combinational logic in FPGA by describing logic equations. The look up table of Virtex5 is a 6 independent input LUT. With two independent outputs O6 and O5. O6 corresponds to the only output of the LUT when it is a six input based function. But when a five input based function is implemented both O6 and O5 can be used as independent outputs as long as both the functions use common inputs. In this case the A1 input of the LUT is tied high. The 6 input LUT is one of the addition in the Virtex 5 logic fabric. The reason for tying the A1 input high can be seen from the figure 3, the A1 input is a select line for the mux which selects the output for the O6 line. This is also shown as an example from a snapshot from FPGA editor in the figure 11. The synthesis tools have the option of combining such functions (enable advanced LUT combining into LUT6_2 Synplify Pro) into a dual output LUT.

Multiplexers (mux): Multiplexers are used to select between the input signals based on the value on the select line. In the MSP430 design multiplexers are used extensively to select between multiple clock lines. Multiplexer implementation on Virtex5 needs the following resources

- 4:1 multiplexers using one LUT
- 8:1 multiplexers using two LUTs
- 16:1 multiplexers using four LUTs

Each LUT can be configured as a 4:1 multiplexer and the output can be registered using a flip flop in the same slice. Each slice has two f7mux the f7amux and the f7bmux

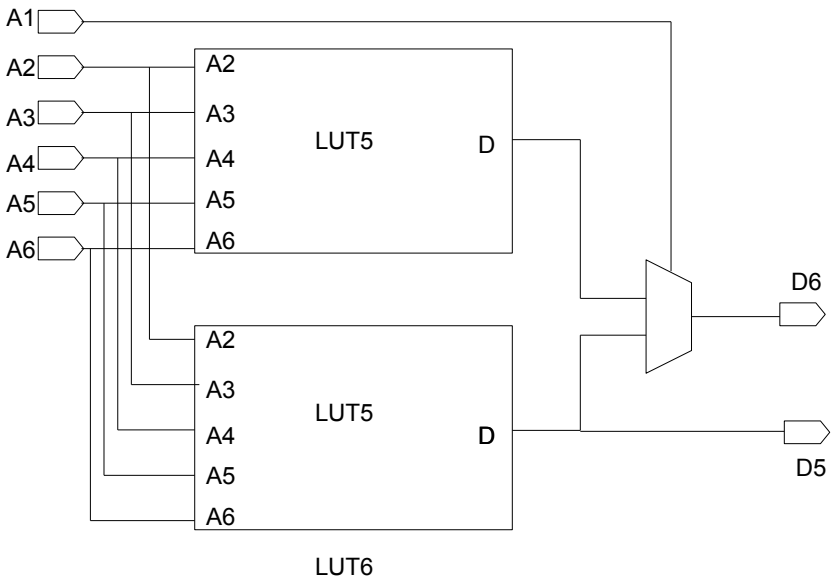


Figure 3: 6 input LUT of Virtex5

which combine the outputs of the 4:1 mux from the LUTs and for a 8:1 mux. Each slice also has a f8mux element that can combine the output from the 8:1 mux to for 16:1 mux. The example of one such implemented f8mux is shown in the figure 4.

Clocking resources: The Virtex5LX110 has 16 clock regions as shown in the figure 5 where each rectangle on the FPGA fabric represents a different clock region. Each clock region consists of 20 CLBs, can have a maximum of 10 global clock lines and spans half a die as shown in figure 5.

Virtex5LX110 has 32 global clock lines which are driven by a global clock buffers(BUFG);

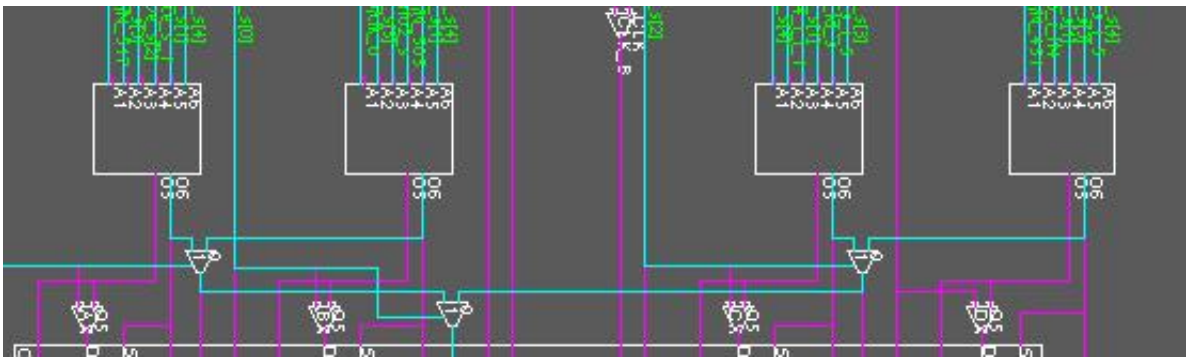


Figure 4: MUXF8 - 8:1 multiplexer.

each half of the die consists 16 bufgs. The BUFGs are situated in the center of the FPGA fabric with access to global routing, this is illustrated in yellow in the figure 5. The global clock lines can drive the clock pin in any of the clock regions of the design. The BUFG is a subset of operation of the BUFGCTRL. The BUFGMUX which is a glitch free clock multiplexer is also a subset of operation of the BUFGCTRL. Further information on the bufgctrl can be found in the Virtex5 data sheet [15]. The bufgmux has 2 clock inputs(I0,I1),one clock output(O) and one select line(S). The BUFGCE has an optional clock enable(CE) line to disable the clock . The main derivatives of the BUFGCTRL primitive are :

Table 5: Global Buffer Primitives

Primitve	**Ports**
BUFG	I , O
BUFGCE	I, O, CE
BUFGMUX	I0, I1, O ,S

In addition for every clock region there are two regional clock buffers(BUFR) and 4 regional clock lines. The BUFRs are capable of :

- Driving three clock regions (one adjacent top and adjacent bottom)
- Clock division
- The CE port can be used to disable the clock output.

Digital clock managers(DCM): The DCMs are part of the clock management tile(CMT) of the Virtex5. Each CMT has 2 DCMs and one pll. For the Virtex5 LX110 there are 12 DCMs and 6 plls. The DCMs are highlighted in red in the figure 5 and it can be seen that they are situated close to the clock buffers in the middle of the FPGA

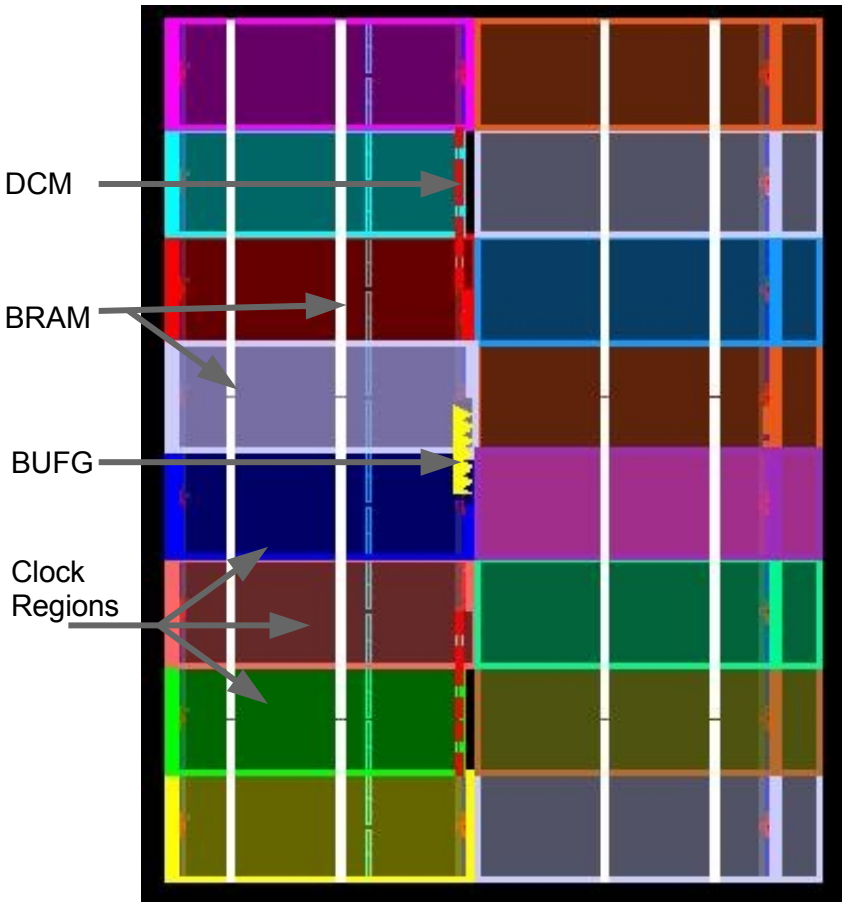


Figure 5: Virtex5 resources

fabric. The DCM functions include clock deskew, frequency synthesis, phase shifting and dynamic reconfiguration. The DCM can be instantiated as DCM_base which is a subset of the DCM_adv primitive.

The relevant ports and attributes for this report is listed below:

- CLKIN: Source clock input.
- CLKFB: Clock feedback input provides a feedback to the DCM to delay compensate the clock outputs and align it with the clock input, this is achieved by essentially a 360 phase shift.
- RST: Reset for the DCM is asynchronous high. Reset must be de-asserted after the source clock has been present and stable for at least 3 clock cycles.
- CLK0 : Provides the same frequency as the DCMs effective input clock frequency.

When CLKFB is connected to CLK0, output is deskewed with respect to CLKIN.

- CLKDV: Provides a clock output that is a fraction of the CLKIN frequency determined by the CLKDV_DIVIDE attribute.
- CLKFX: Provides a clock output that is a function of (M/D)CLKIN frequency. M is the multiplier specified by CLKFX_MULTIPLY attribute and D is the divisor specified by the CLKFX_DIVIDE attribute.

The DCM has DLL and DFS outputs. Along with this there are max speed and max range attributes. Based on these factors the input, output and feedback clock frequencies have to be selected. An attempt to explain these limitations is explained in the figure 6.

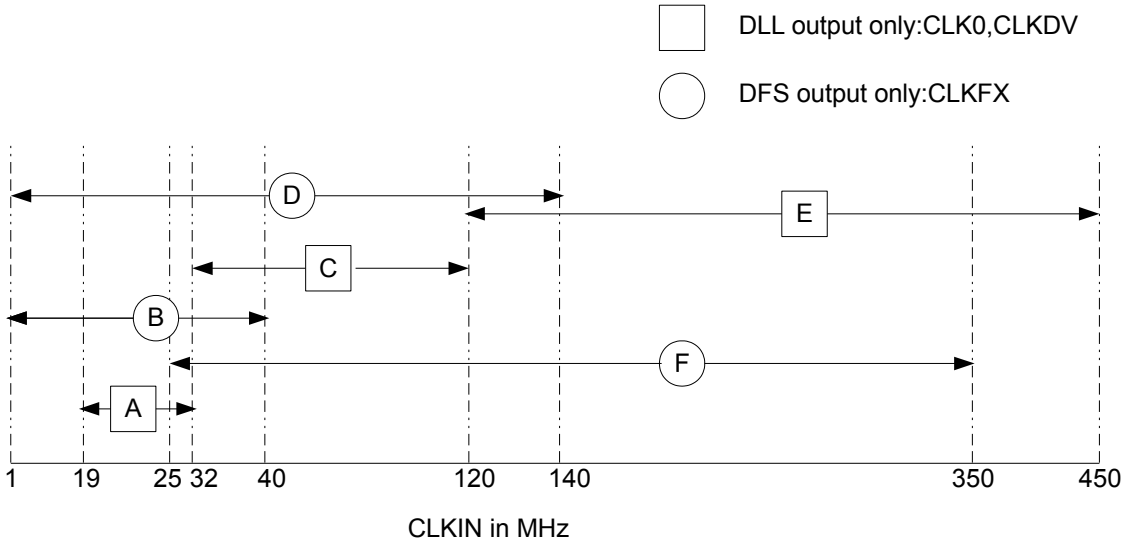


Figure 6: DCM operating range

The configuration for the regions are described:

Table 6: Output frequency limits

Region	clkfx min	clkfx max	clk0 min	clk0 max	clkdv min	clkdv max
A,B	19	40	19	32	1.19	21.34
D	32	140	32	120	2	80

- Region A: Maximum Range mode with low frequency mode of operation using DLL outputs only.
- Region B: Maximum Range mode with low frequency mode of operation using DFS outputs only.
- Region D: Maximum Speed mode with low frequency mode of operation using DFS output only.

When the DCM is set to maximum speed mode it is optimised to produced high frequency clocks with low jitter. When it is set to maximum range mode it is optimised to produce low frequency clocks with a wider phase shift range[15]. The figure 6 illustrates regions A to F. Some of the limitations while using the DCM for our design are:
The input clock frequency is 24 MHz. Hence only regions A,B and D can be used. This assumption is based on the fact that the input to the DCM is an external 24 Mhz source. It can also be the case that DCMs can be cascaded to obtain better flexibility, however cascading of DCM is discouraged by Xilinx unless it is absolutely necessary as described in the answer database 18181[3].

While using the DCM for clock manipulation the Xilinx directive is to use the CLK0 as a feedback to the CLKFB pin to avoid any skews on the output clock. If CLK0 is fed back to CLKFB, the CLKFX outputs are phase aligned to CLKIN whenever that is mathematically possible. Since the CLK0 is a DLL output, only region A with DLL output can be used.

In case higher frequencies like 140 MHz are necessary , region D needs to be used, which is a DFS output only (CLKFX output). But in this region the clock feedback cannot be used as seen from the table 6.

Multiplication and division are performed as a combined mathematical operation. Assume FIN = 50 MHz, M = 25, and D = 8. In this case, CLKFX is then 156.25 MHz, even though FIN x 25 = 1.25 GHz, which is well above the maximum frequency of 550 MHz.

Block RAMs(BRAM): The Virtex5 block rams store 36kb of data. The BRAMs are highlighted as columns of white lines in the figure 5. There are two such columns on each half of the Virtex5 accounting for 128 BRAMs capable of a total of 4Mbits of memory. There are different aspect ratios for the data and address bus that are possible right from 36x1,18x2 and so on. The read and write operations are synchronous. BRAMs are capable of functioning as dual port rams. The two ports are independent of each other sharing only the stored data on the BRAM. Read and write operations require one clock edge.

BRAM initialization By default BRAM memory is initialized with all zeros during device configuration. For the ramb36 primitive the 128 BRAM initialization attributes range from init_00 to init_7f. The init_yy attribute can be decoded using the following formula
from [(yy + 1)x 256] 1 to (yy)x 256

For example, for the attribute INIT_1F, the conversion is as follows:
yy = conversion hex-encoded to decimal (xx) 1F = 31
from [(31+1) x 256] 1 = 8191 to 31x256 = 7936

For example if we want to program a memory location at address 130; the right init_xx attribute from the above equation has to be used.

The location init_10 corresponds to the address range 128-135. In the initialization strings the least significant bit is at the rightmost position. The init attribute can be used either in VHDL,Verilog,ucf , PlanAhead or even FPGA editor. For the implementation flow in the project, the BRAMs were initialized using scripts in the FPGA editor.

6 ZeBu to FPGA recoding

6.1 ZeBu introduction:

ZeBu is a Virtex4 based ASIC prototyping platform capable of 0.75M ASIC gates(for the smallest system) from Eve. ZeBu functionality includes hardware debugging, cosimulation and software development. Input for ZeBu includes EDIF netlist produced by standard synthesis tool like Synplify Pro and XST. The ZeBu compiler takes care of clock handling and memory handling which are all black boxes in the EDIF netlist produced by the synthesis tools.

Memory modeling in ZeBu The memory required for modeling the DUT is realized either as FPGA BRAM memory or as ZeBu embedded memory.

Small size memories (from a few bits to 1Kbit) based on Virtex4 distributed ram these can be inferred by synthesis tools or instantiated by IP core generators. Distributed RAMs are emulated on LUTs and hence consume FPGA logic, moreover the distributed RAM are not dense.

Medium sized memories (form 1kb to 500kb) based on Virtex4 BRAM these can be created by inferring from synthesis tool or ip core model.

Large memories based on on board DRAM and SSRAM memory chips.

ZeBu supports multi port memories from 8 to 32 ports based on Virtex4 FPGA. Medium size memory models can be synchronous or asynchronous.

Clock handling: For clock handling, the ZeBu separates the clocks into primary and derived clocks. Primary clocks are the main input clocks to the design. Primary clocks are declared in the dve file. Derived clocks are obtained from the primary clocks and refer to divided clocks,gated clocks and multiplexed clocks. The problem of using such derived clocks in an FPGA design is the resulting skew due to the use of local routing resources. This resulting skew is automatically fixed by using skew adjustment(essentially delay insertion) from the zCui interface.

Debugging: The debugging feature in ZeBu is provided in terms of static and dynamic probes. Dynamic probes provide access to sequential signals during runtime and do not affect the design and the design does not need a recompilation. Static probes on the other hand need to be declared in the dve file and addition of a new static probes indicates a design recompilation.

6.2 Implementation overview

The flow for the FPGA conversion is shown in the figure 7. The implementation process passes through several phases before the final bit configuration file is generated. Each stage is explained in brief in the following paragraphs.

Synthesis: It is the procedure of building a EDIF[3] netlist from the input RTL which might be described in Verilog, VHDL or a mix of the both the hardware description

[3]Electronic Design Interchange Format is a neutral format to exchange electronic design data. Its primary use is CAD to CAD data transfer format. It conveys electronic netlists and schematics.[7]

languages.

NGD build: This process reads the EDIF netlist and creates a Xilinx native generic description file - NGD. The NGD file contains the details of the design in terms of logic primitives such as logic gates, rams as well as the design hierarchy present in the input file. The steps involved in the NGDbuild process include:

- Reading the source netlist
- Reduce all components in the design to NGD primitives
- Check the design by running logical design rule check[4]
- Write the NGD output file[9]

Input to the NGDbuild is a EDIF netlist file and output is a NGD file.

MAP: The map programs maps the logic design to Xilinx FPGA. The input file to the map process is a NGD file and the output is a NCD[5] which is a physical representation of the design in terms of Xilinx primitives[9].

Place and route (PAR): PAR places and routes the design. PAR is executed in placing and routing phases.

Placing The placer places components into sites based on factors such as constraints in the pcf file, length of connections and routing resources available. Placer writes an output NCD file.

Routing It performs the interconnection between the placed logic components.The routing is run through multiple phases where it tries to route the design to meet the design timing constraints. The router writes a NCD file every time it completes a phase[9].

[4]performs 6 rule checks to verify that the design meets these rules.For more information refer to the DRC paragraph on page 46

[5]native circuit description

TRCE: TRCE provides a static timing analysis of an FPGA design based on input timing constraints. It performs two main functions:
Timing verification: checks the compliance of the design with the timing constraints.
Reporting: Lists all the paths that have passes or failed the timing verification. The TRCE can be run on unplaced design, partially placed design, fully placed design, partially placed and routed design, completely placed and routed design[9]. The input files for TRCE is NCD file and/or pcf[6] file and the output is a twx/twr result file.

Bitgen: Bitgen generates a bitstream to configure the Xilinx device. The bit files contains the configuration information from the NCD file. The input NCD file contains information on internal logic and the interconnection. The bit file is downloaded into the memory cells of the FPGA device which configures the device. Bitgen takes a NCD file and produces a configuration bitstream file (.bit)[9]. The bit stream generation for our flow is performed in the FPGA editor. It can also be performed in the PlanAhead environment.

The overview presented in the above paragraphs are elaborated in the next few chapters.

Clock Block: As discussed earlier in the section 3 the MSP430 has upto 5 clock sources. Each of these clock sources are emulated using a DCM. The input clock to the DCM was an external 24Mhz source. For more information on the Virtex5 DCM please refer to the section on DCM on page 11.

The clock block for the MSP430 design was modeled based on the RTL presented in appendix A.3. The DCM configuration needs to follow certain guidelines for proper operation and these are listed here:

- The reset of the DCM can only be released after the input clock is active and stable. This is achieved by using a counter that counts upto 100 cycles of the input clock before deasserting the reset pin on all the DCMs.
- The output clock of the DCM must pass through a BUFG before it sources the clock pin of any of the FPGA primitives.
- Component switching limits must be observed by configuring the DCMs in the right mode of operation.
- CLK0 output must be fed-back to the CLKFB input through a BUFG to reduce clock jitter.

[6]physical constraint file

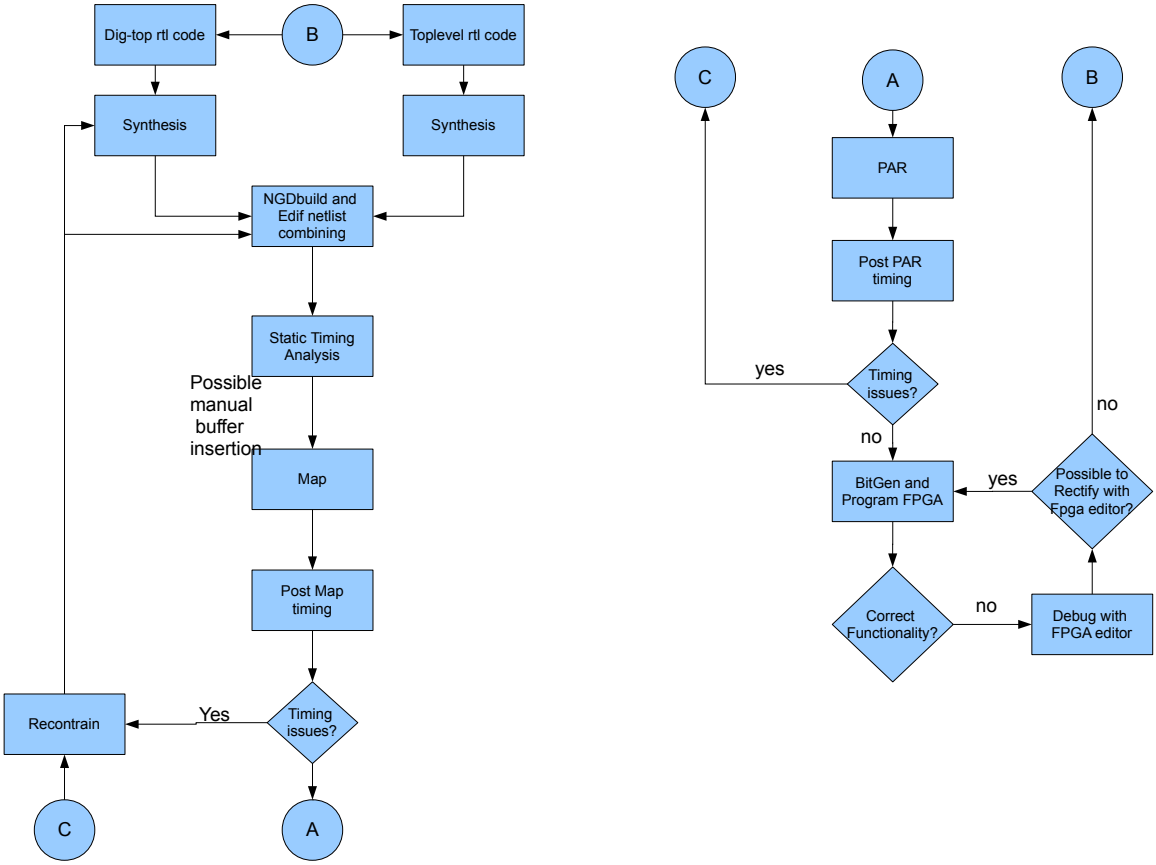


Figure 7: Flowchart

Virtex5 DCMs have a DLL and a DFS output. The DLL and DFS can operate in either low or high frequency mode of operation. In addition there are max speed and max range of operations. Each combination of the above modes of operation present different limits of operations for the input and output frequency. As an example one of the clocks of the MSP430 requires a frequency of about 150 MHz. This clock is used to source the delay shift registers and also in the process of modeling the analog functionality. With this mode of operation it is not possible to have a feedback on the DCM and this can be illustrated in the figure 6 and table 6.

In case the DCM are not operating within the limits there are no warnings during simulation, synthesis or during NGDbuild. But there are warnings in the MAP phase that indicate that there is a problem with the timing constraint. Even in this case the problem cant be recognised right away, only after running a timing analysis using the timing analyzer can the problem be identified as being related to component switching limit. Hence, if precautions are taken right at the outset, many of these issues can be

by-passed.

Global signals: VHDL signals are used as interconnect and convey information between components. the scope of signals in VHDL is described in the following table

Table 7: VHDL signal scope

Declared at	**Scope**	**Comments**
Package	Global	Can be used globally by including the package declaration
Entity	Local to the entity	Can be used in any architecture definition inside the entity
Architecture	Local to architecture	Can be used only inside the current architecture

The MSP430 design requires signals that are accessed deep in the hierarchy of the design. These signals are mainly clock signals. Declaration of global signals in these packages eliminates the need to add additional ports on all the modules that require access to these signals. The RTL code used for usage of global signals are provided in the appendix A.1 and A.2. It can also be observed the clock signals are made global in the clock block just by including the packages in the code appendix A.3. After synthesis the following fan out on the global signals were observed.

Table 8: Global signals fanout

Signal	**Fan out**
zxlsys	236
pgen	41
dcoclk	9
modosc	2
pssclk	3

Based on these results , usage of global signals reduces effort and prevents the possibility of human error involved in port changes to the design.

Memory: The MSP430 has RAM and flash memory. These ASIC memories are emulated on the FPGA BRAMs. The aspect ratios of the memories used in the MSP430 are given in the table below along with the number of BRAMs used to realise these memories.

Syncore IP generator has a graphical user interface that enables the user to customize Synplify cores. This was the method used to instantiate BRAMs for the RTL. The RTL thus generated is provided in the appendix A.4. The Verilog file generated can be used

Table 9: MSP430 memories

ASIC memory	**memory size(in bits)**	**BRAMs used**
ram	16x1024	1
info flash	32x1024	1
main flash	32 x8192	8

to infer the required memory size of the BRAM by just changing ADDRWIDTH parameter and the bus width can be changed by changing the ADDRWIDTH parameter. For example the code in appendix A.4 is 32x1024 bit memory component, in case if the capacity has to be increased to 32x4096 bits ; this can be achieved by changing the ADDRWIDTH to 12. Thus, in order to change the size of the BRAM does not require a fresh instantiation using the GUI everytime. A Verilog wrapper has to be written to interface this segment of code to our actual design. A snapshot of the GUI along with the options are shown in the figure 8. The implementation of the mainflash is interest-

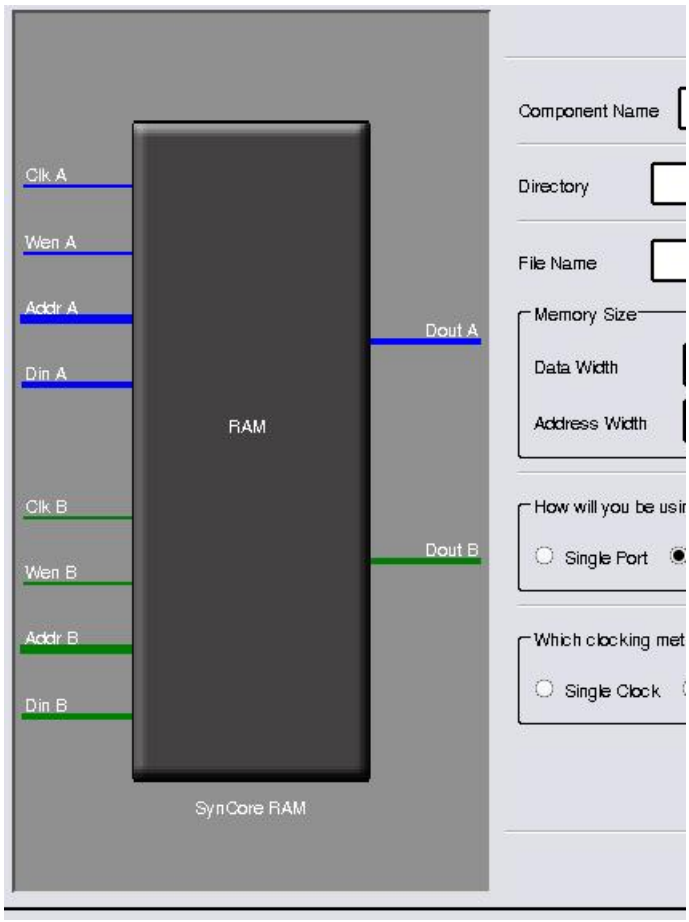


Figure 8: Syncore memories

ing to look at. It uses 8 BRAM modules . The implementation for the mainflash can

be explained using the table 10. Each BRAM had 13 bit address input, but the data bus on each of the 8 BRAMs was just 4 bits wide and 1024 bits deep. So essentially, this implementation is of eight 4k BRAMs. There are some limitations in customizing the cores through the interface, the MSP430 ASIC memories require a read operation on the positive edge and a write operation on the negative edge. This feature was not provided while instantiating the BRAM.

The flash can also be inferred through RTL, in general the basis for ip and inference is the same as explained in the paragraph on page 23.

The memory requirement for newer devices of the MSP430 are as high as 1MB. The Virtex5 has just 128 BRAMs which means that a maximum 4Mb memory can be achieved(overlooking other practical issues that might arise due to BRAMs scattered on the FPGA fabric). So, a general case for the maximum achievable size was tried. For achieving a memory size of 32x32kb the address was 15 bits wide and 32 blocks of BRAMs were used with each BRAM having one data line connected to it. So, far the usage of BRAMs has been intuitive. In case of implementation of 32x64kb RAM, BRAMs need to be cascaded together to achieve this case. These results are generalised in table 10.

Table 10: BRAM aspect ratios[15]

Port data width	**Port Address width**	**Memory depth in bits**	**Addr bus**
1	15	32768	14:0
2	14	16384	14:1
4	13	8192	14:2
9	12	4096	14:3
18	11	2048	14:4
36	10	1024	14:5
1(cascade)	16	65536	15:0

In case of usage of cascadable block rams, the address width is 16 bits, unlike the cases observed before. So, from this implementation it can seen that the highest bit selects between the two cascaded BRAMs as every bit x of the data line connects to two BRAMs mem_mem_0_x and mem_mem_1_x. So, when the 16th address line is low it selects the data bit x to be written to mem_mem_0_x and when it is high it is selected to be written to mem_mem_1_x. The Virtex-5 user guide suggests that when the BRAMs are not cascaded the 16th bit has to be connected high, but the syncore implementation has this bit grounded and the components are named mem_mem_0_n where n is the number of data bits.

In addition to BRAMs, FPGA offers one more type of memory element. The LUT in the slicem can behave as storage elements, these are however inefficient and not dense for larger memories.Usage of distributed rams will also lead to higher utilisation of the FPGA resources.

Manual insertion of buffers: As seen in the section 3 on the MSP430 introduction, the ucs has upto 5 clock inputs and 3 clock outputs. In addition each of these clock outputs can be divided to give a fractional frequency output. These multiple clocks mean that the clock structure for the MSP430 is complex. In an ASIC fabrication the clock tree can be customised to provide low skew routing. However, in the FPGA due to limited low skew routing resources some generated clocks might be subjected to large magnitudes of clock skew. These clock skews may introduce hold violations as discussed in section 8. In some cases to overcome hold time violations it was necessary to manually instantiate buffers to boost a clock signal or to delay a data path eventhough this is not a recommended design practice. One such example is discussed in the paragraph regarding ifclk on page 55. The output of this clock is a multiplexed clock. Moreover the fan out of this clock is 3316, to minimise the clock skew a bufg could be inserted after the mux or even better a bufgmux primitive could be instantiated in place of the mux.

7 Synthesis

Synthesis tools are used to produce an EDIF netlist from the input RTL files. The vendor specific netlist and constraints are obtained after logical mapping and optimisation which mainly constitute the synthesis process. The optimisation is covered in section 7.2.

The synthesis tools used for this thesis was Synplify Pro and the versions include 2009.06-SP1-p, 2010.03, 2010.03-sp1, 2010.09-beta. However there are other FPGA synthesis tools which include Leonardo-Spectrum/Precision RTL from mentor graphics, XST from Xilinx, Quartus from Altera.

7.1 Inferring vs Instantiation

Digital functions in the RTL might need to be mapped to specialised blocks on the FPGA such as BRAMs, DSP48, SRL. In such cases the user has the option of manually

instantiating these blocks or inferring them with the help of the synthesis tools. The difference and similarity between these two approaches is discussed in the following paragraphs taking the example of IP[7] blocks.

Synplify Premier has DesignWare building block IP which is the IP repository for the Synplify environment. It consists of IP ranging from basic functions (common arithmetic and logic functions) to DSP, memory, interface and application specific IP blocks.

You can use the IP blocks in your synthesis by either operator inference or component instantiation. For example consider the flow of synthesis in Synplify as shown in figure 9. When an addition operation is indicated by a plus sign, the tool through operator inference selects a synthetic operator [8]. Therefore, operator inference translates the HDL operator to a matching synthetic operator. This synthetic operator may have several actual implementation options in the design library which are selected based on the optimization criteria. A particular implementation can also be forced by using the implementation option[5]. In the given example a constraint of timing results in carry look ahead implementation and an area constraint causes a ripple carry implementation of the adder. In order to make this decision of implementation the tool creates a pre optimised model for the possible implementations. The timing and area characteristics serve as basis for implementation selection[5].

Similarly Xilinx has Core generator tools for IP cores[11].

Component instantiation is performed by explicitly instantiating the IP like any other component. As in the case of operator inference, the tool finds the appropriate synthetic module for the instantiation and chooses from all the associated implementations of the synthetic module.

7.2 Optimisation by Synplify Pro

In the process of producing the final EDIF netlist from the RTL code the synthesis tool performs several logic optimisation on the input design. Some of the optimisations carried out by Synplify Pro are described in the following paragraphs.

Arithmetic Optimization which use expression rearranging for optimization.

[7]Intellectual Property

[8]Represents the operation which is called for by the HDL operator

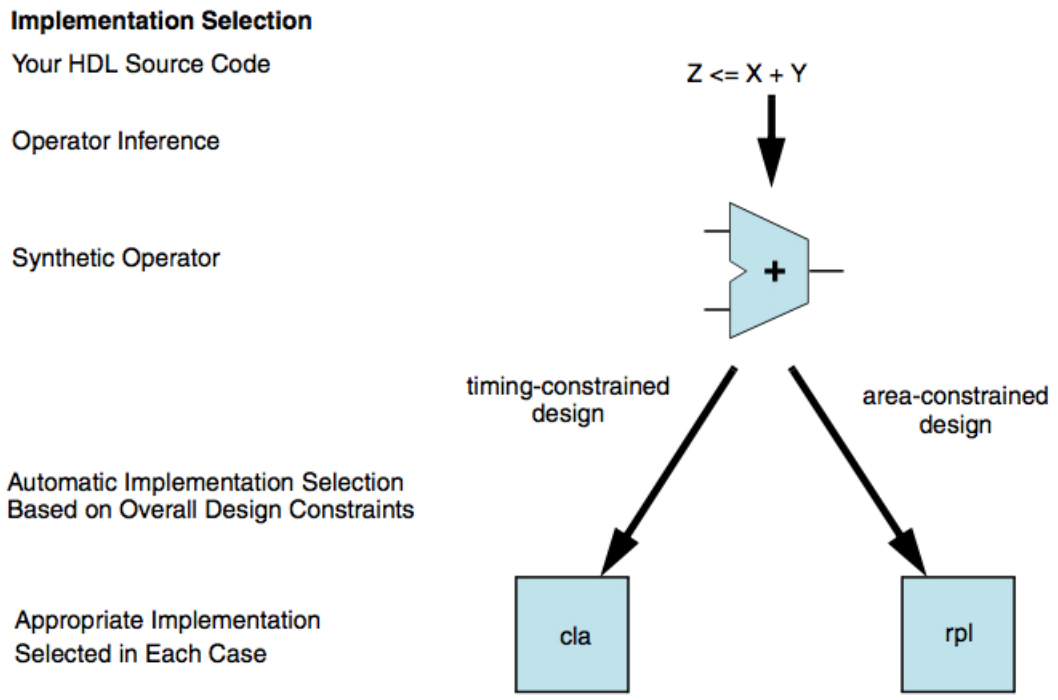


Figure 9: Synthesis Flow [5]

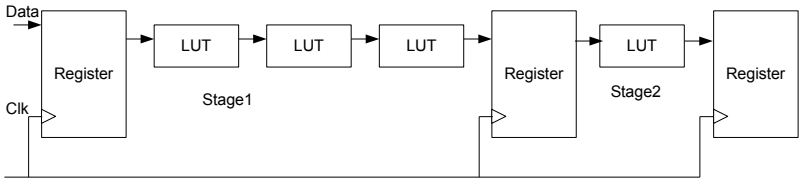
Resource Sharing uses the same computational blocks for similar operations that do not occur at the same time.

Pin Permutation is used in operations where changing the order of inputs does not affect the operation.

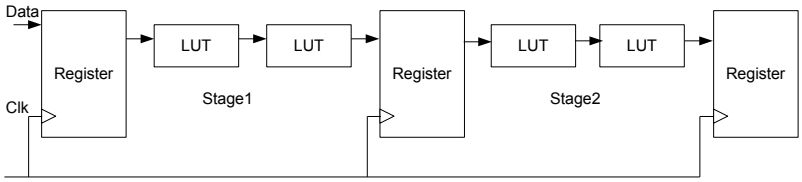
Retiming is a tool option. This can improve the performance of sequential circuits without having to modify source code. This is achieved by moving registers in a sequential path. A simple example of retiming in shown in the figures 10a and 10b.

Before retiming the slack available for stage1 is much less than the slack available for stage2, but by rearranging the registers after retiming the extra slack available on stage 2 can be used to accommodate an additional combinatorial logic.

Register Duplication Is automatically carried out by the implementation tool to reduce delay in high fanout paths. This is carried out even if path meets fanout guide (the numerical value set under this option for Synplify Pro is for all nets in the design.



(a) Before Retiming



(b) After Retiming

Figure 10: Retiming

The tool uses this as a guide and tries to ensure that all nets have fanouts that are under this limit [6]).

Advanced LUT combining As discussed in the section about LUT on page 9 each Virtex5 LUT are capable of two independent outputs provided the functions producing these outputs have a common input. With this option enabled Synplify Pro combines

common input functions into compatible LUTs. The figure 11 illustrates an actual synthesis of such an "LUT6_2". The lines in blue represent the active connections. Hence both the O6 and O5 outputs are active. Similarly it can also observed that the common inputs for both the functions are A3 and A2 only.

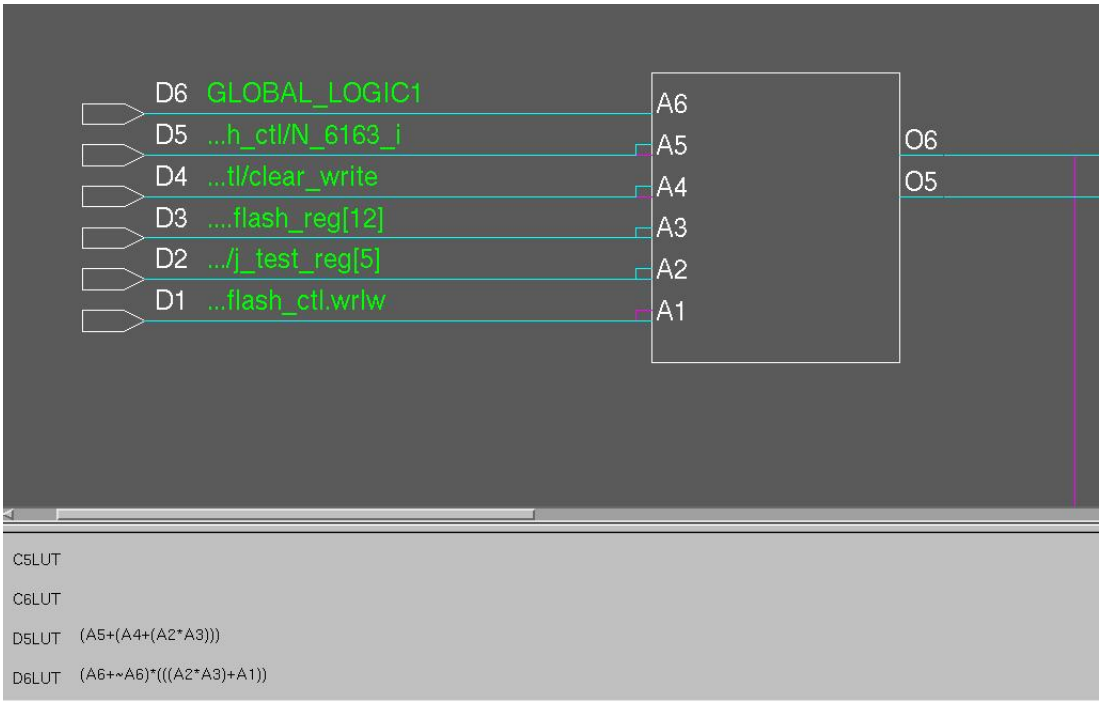


Figure 11: Dual output LUT

The synthesis result is presented in the table 11 which gives an overview of LUT utilisation for the MSP430f513x design with the advanced LUT combining option enabled. The total number of LUT used for the design is 16097.

Table 11: LUT utilisation

LUTs realised	**total number**	**Percentage**
6 output LUT	13599	84
5 output LUT	56	1
dual output LUT	2406	15

Generated clock conversion Generated clocks are produced from other clocks in the design with logic such as division circuit. The fix generated clock option tries to

use the parent clock as the clock for all the flip flops that use the generated clock and an enable signal to emulate the behavior of the generated clock.

The requirement for generated clock conversion are:

- The combinational logic must be driven by flip flops.
- The input flip flop cannot have active set or reset.
- All input flip flops must be driven by the same edge of the same clock [6].

Generated clocks include clock division by ripple counter technique. This can be implemented in the PLLs, instead of implementing it on registers. But since the philosophy of the implementation was as little change as possible, this was not continued.

IOB packing The Virtex5 input and output logic resources have the basic capability of edge triggered D type flip flop and level sensitive latch. Input/Output Blocks Packing refers to placing registers in IOB close to the pad. With this option only the first flip-flop or latch encountered in the path from a pad can be moved into the IOB.

This moves the flip flop or latch closer to the pad, so the pad to setup time decreases but the delay from IOB to flip flop or latch to the next synchronous element increases. Thus reduction in the setup time could lead to increased frequency of clock[18]

The need for iob packing is most essential when

- The chip interfaces with another, and you have to minimise the register to output or register to input delay.
- The design has limited CLB resources thus packing the registers in an iob can free up some resources.

For the MSP430 design neither of the condition exists but this was enabled as an experimental case to look at the implementation results. The figure 12 shows the register on the pin pj3 that has been packed into an ilogic cell.

Gated clock conversion and constraints Clock trees dissipate about 50% of the dynamic power in a design. So, the use of gated clock can minimise this power dissipation. In gated clocks the clock is selectively enabled or disabled based on the enable

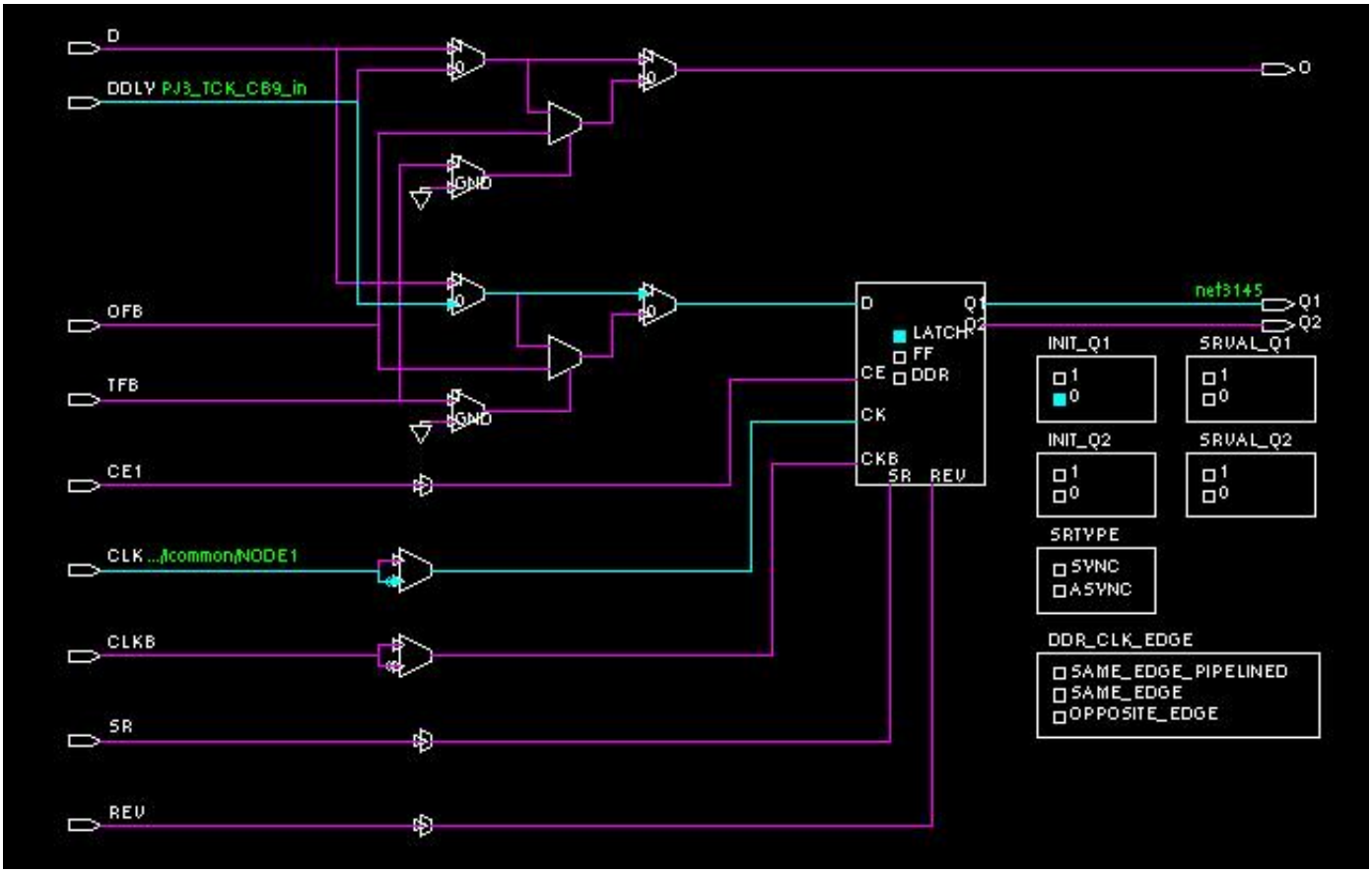


Figure 12: IOB packing

signal. However the use of gated clocks in an FPGA design has the following disadvantages:

- Using non dedicated routing for gated clock signals causes routing congestion
- Routing congestions may lead to longer par runtimes
- Use of non dedicated routing for clock signals lead to clock skew which may lead to possible hold time violations

The conditions necessary for gated clock conversion are

- The gated logic must consist of combinational logic only, a derived clock that is the output of a register is not converted.
- There has to be only one input to the combinational gated logic which acts as a base clock.
- Correct logic format- the combinational gate must satisfy the following conditions:
  - For at least one set of gating input values, the output for the gated clock must be constant and not change as the base clock changes.
  - For at least one value of the base, changes in the gating input must not change the value of the output of the gated clock.

With these ideas as a foundation , the following sections provide a discussion of the problems encountered in the gated clock conversion of MSP430.

Selective inversion and muxed clocks Selective inversion of clock is achieved by using an xor gate. But the use of an xor gate does not satisfy the correct logic format specified in the previous paragraph. So the selective clock inversion by the xor operation cannot be converted to one base clock.

Multiplexed clocks are used when the circuit has to be run at multiple clock frequency. The figure 13b shows a 2:1 mux which selects between two clock signals. In such cases a global buffer has to be added at the output of the mux, this is further discussed in the section manual insertion of buffers.

Further, the output of the mux might pass through a gating logic, in which case the tool has to recognise a base clock for the gated clock conversion. When a multiplexed clock is present in the design, the error message for the gated clock conversion is"multiple declared clocks found" since the synthesis tool cannot interpret that only one of the

input clocks is switched to the output at a time. The design with multiplexed clock has to be properly constrained to overcome these issues. The following examples give an overview of how this is possible[2].

In the first example based on figure 13a the 2:1 mux generates a clock output based on two exclusive input clocks. The constraints for such a muxed clock is straight forward. The following constraints can be used for it.

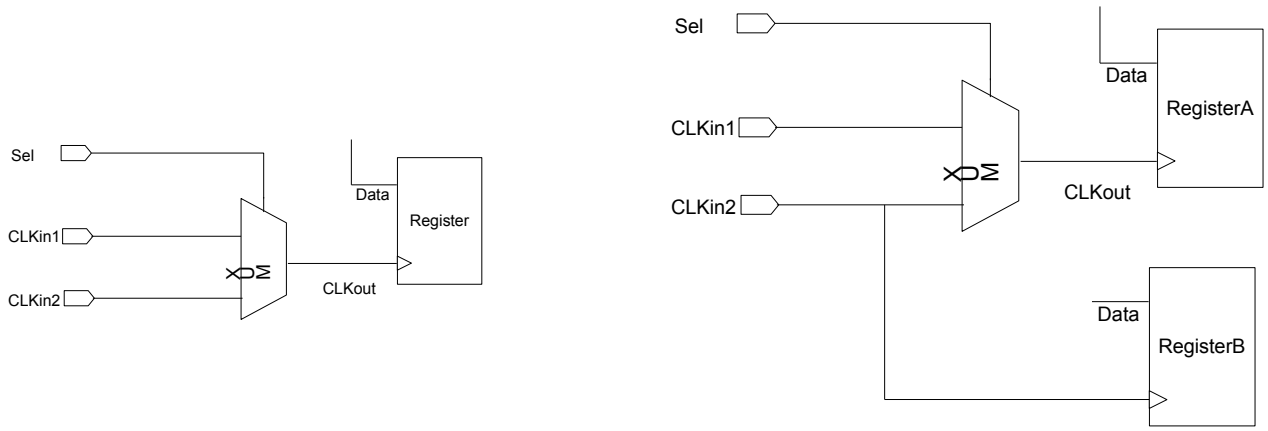
```
create_clock [get_ports {CLKin1}]
-name clk1
create_clock [get_ports {CLKin2}]
-name clk2
set_clock_groups
-exclusive
-group clk1
-group clk2
```

However, in the MSP430 design the clock signals at the input of the mux may not be mutually exclusive. In this case the example of the linked clock mux as shown in figure 13b can be used. For this case the following constraints might be used to indicate to the tool that the output of the mux are exclusive to each other i.e one only clock is switched in at a time.

```
create_clock [get_ports {CLKin1}]
-name clk1
create_clock [get_ports {CLKin2}]
-name clk2
create_generated_clock [get_pins{CLKout}]
-name clkout1
-source [get_ports {CLKin1}]
create_generated_clock [get_pins{CLKout}]
-name clkout2
-add
-source [get_ports {CLKin2}]
set_clock_groups
-exclusive
-group clkout1
-group clkout2
```

This example creates two clock groups on the output of the mux with each clock group corresponding to the input clocks, this is done with the -add directive. So here the

output clocks are defined as exclusive and false path relations of the output clocks with the input clocks and its relation with other timegroups need not be manually specified.



(a) Muxclk with exclusive inputs

(b) Muxclk with non-exclusive inputs

Figure 13: Muxed Clocks

A physical implication of muxed clock is the presence of glitches when the clocks are switched from one clock source to another. This is overcome in the MSP430 design by using these rules during the switching of the clocks

- The current clock cycle continues until the next rising edge.

- The clock remains high until the next rising edge of the new clock.
- The new clock source is selected and it continues operation[8].

For more information please refer to the user guide of the MSP430F5xx family[8].

Describing base clock: The output clock at an xor gate or a muxed clock have to be defined as a base clock for further gated clock conversion in the design. The method that was initially used in the design was defining the base clock after every selective inversion and muxed clock conversion.

Locking pins with xc_loc constraint: The Virtex5 has 1760 IO pins but only few of these pins are available on the falcon board. So, in order to have all the port pins of the MSP430 the IO pins have to be locked to accessible FPGA pins.

This IO locking can be performed either in the RTL code or in the ucf input file to the PlanAhead environment. In the RTL the IO pin lock can be achieved using the xc_loc attribute. The possibility of locking the IO pins in the PlanAhead tool is discussed in the next chapter.

Control sets In an FPGA control set refers to each unique combination of clock, clock enable and synchronous set/reset signals. Only registers that use common control sets can be packed into the same slice. This observation is significant because the Virtex5 has 4 registers in a slice. Hence, registers not sharing the same control set might lead to poor device utilization density. The MSP430 has 1653 unique control sets, so for the implementation of future devices if there are issues with density of device utilisation the unique control sets can be minimised by using the syn_reduce_controlset_size constraint.

The above directive moves some or all of the control inputs of the register to the data input. By default the tool chooses right number of control sets based on qor, resource available and target technology.

However, for our current design there are many slices where there are 100% register utilization as shown in the figure 14. The figure illustrates the flip flop utilisation per CLB. Red indicates utilisation between 80 to 100% and the yellow squares indicate 60-80 percent utilisation. Moreover, the device utilisation is 24%, so no congestion for resources should exist. Hence this is not a demanding constraint.

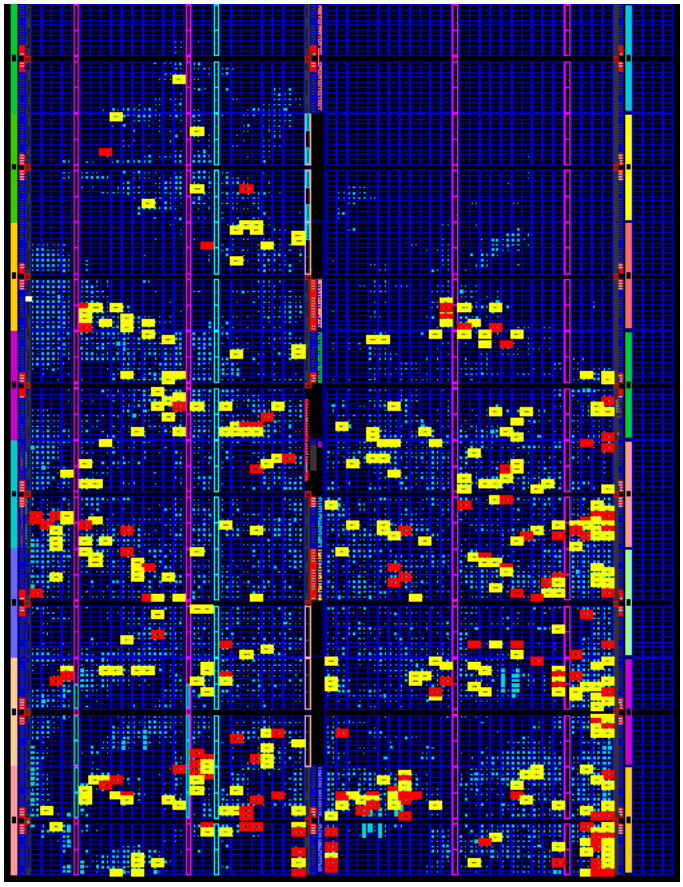
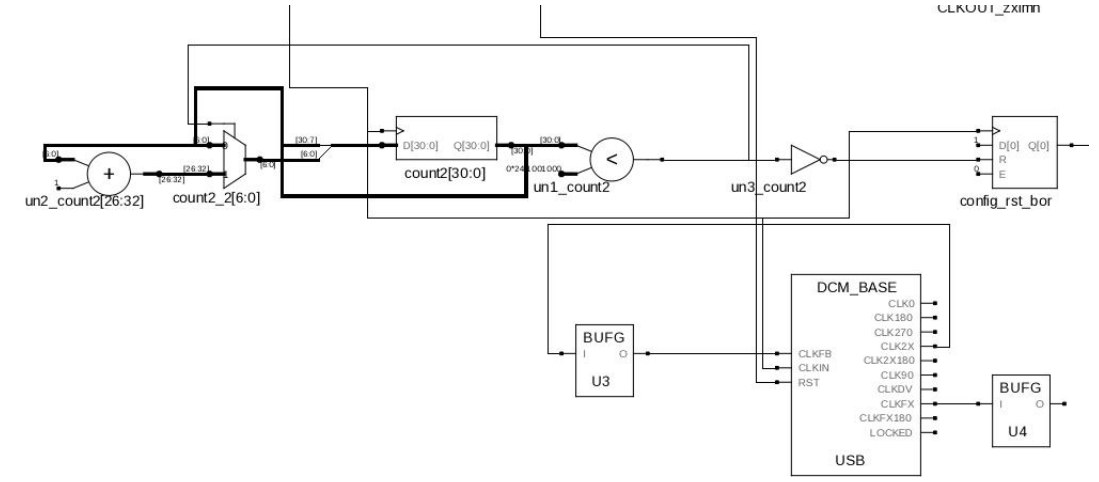


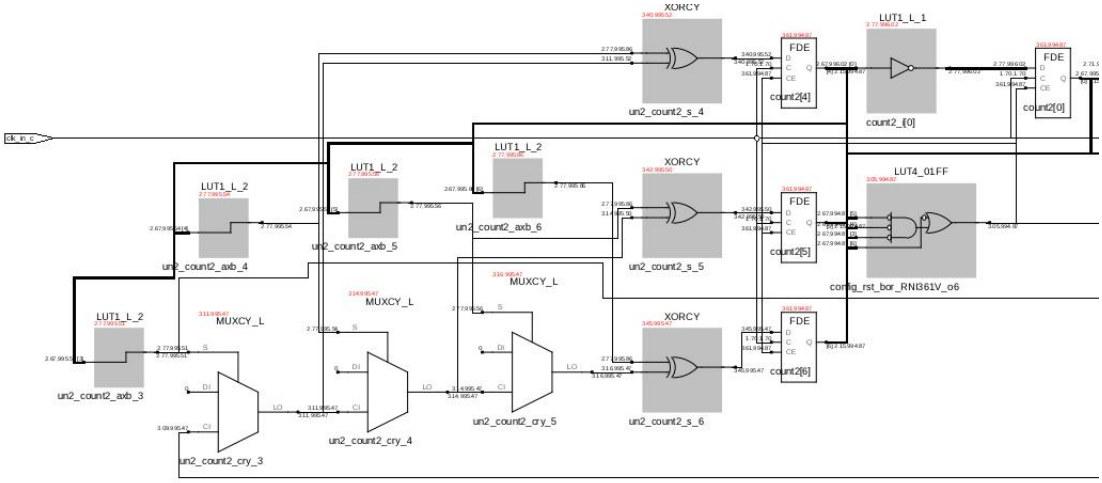
Figure 14: Flip Flop utilisation per CLB

Synthesis output files: At the end of the synthesis cycle, Synplify Pro provides us with a EDIF netlist, a ucf constraint file along with an RTL and technology view. The RTL view provides a high-level (components not mapped to FPGA LUTs, registers or FPGA specific resources) technology independent view after the compilation of the RTL. The RTL view has the advantage of representing the RTL code as it is in terms of non FPGA primitives. This enables the designer to have a visual hierarchical overview of the design while debugging. On the other hand technology view provides a low level, technology specific view of the design after synthesis. The technology view has the RTL mapped to FPGA building blocks. The usefulness of this view is discussed in the probes section of the FPGA editor on page 52. The figure 15a and 15b shows the example of a counter in a RTL view and technology view respectively. The RTL shows the counter as a high level component but in the technology view it is mapped to fast lookahead carry logic of the FPGA.

Synthesis issues Some synthesis issues were faced during the RTL synthesis on Virtex5, but surprisingly this was absent during Virtex4 synthesis. The synthesis of the



(a) RTL view



(b) Technology view

Figure 15: Synplify Pro schematics

design never completed with gated and generated clock conversion enabled. There was however no problem when the design was synthesized without constraints. However, this workaround was not feasible. In addition, the synthesis did not complete with the design as one entity as a result the design was split into two existing partitions the dig-top and the toplevel , this is also depicted in the figure 7.

Synopsys were able to reproduce this issue and this was a result of a runtime bug. The root cause was identified as an infinite loop in incremental timing update of the Xilinx Virtex5 mapper[9]. The software bug was fixed in the map520rcp1branch of the Xilinx mapper[4]. The bug fix was finally integrated in to the 2010.09 beta release of the Synplify Pro tool.

To overcome this issue the design was partitioned into top level and dig-top for the synthesis to complete. The idea was to later combine the EDIF netlists in the Xilinx environmnet before the map stage.

7.3 EDIF netlist combining

Since the dig-top does not have any external port and it is combined into the top level, the automatic IO insertion had to be disabled. In the absence of this option pads are created for the interface between dig-top and top level and results in the following error messages.

In case of input ios being present the resulting error message is
ERROR:NGDBuild:455 - logical net 'p1_set_ifg_2v_s[6]' has multiple driver(s): pin PAD on block p1_set_ifg_2v_s[6] with type PAD, pin Q on block u_5v_wakeup_logic/ u_wakeup_logic_P16/WKUP/DTC0/TI_LACTHC_UDP3/q with type LDC

In case of output ios being present the error message is
ERROR:NGDBuild:809 - output pad net 'p1_clr_idet_2v_s[6]' has an illegal load: pin I1 on block u_5v_wakeup_logic/u_wakeup_logic_P16/WKUP/LAN1/Q_RNO with type LUT2

When an EDIF netlist is used as a submodule of another design, the following conditions must be met

[9]Incremental timing update is an iterative analysis during the mapping phase. Based on this information the tool is trying different kinds of optimizations on a RTL design. The static timing analyzer will take the result of that analysis and use this for different reports.

- In case the ports of the sub-module are directly connected to the top module, the ibufs/obufs can be used in the sub module
- But in case if the ports of the submodule are not directly connected to the top-level the ibufs/obufs must not be inserted in the top module.[6]

8 Timing Constraints

Timing constraints are essential to convey to the synthesis and implementation tools the requirement of the design in terms of frequency of operation of various clocks. The tools take these constraints as guidelines and try to generate a resulting implementation that complies with the input timing constraints. The timing constraints for the Synplify Pro tool are in the form of sdc file format and for Xilinx tools in the form of ucf and pcf file formats. In addition the implementation tools provide us with timing check features that can be used to verify if the implemented design meets the constraints. These tools and principles used in timing analysis is discussed in this section.

Static timing analysis: The timing analysis tool from Xilinx has the following features

- Static timing analysis on FPGA designs
- Timing analysis post map, post place or post route of FPGA design
- It performs setup and hold checks. It also checks for component switching limits
- It reports the delay along a given path or paths and reports the slack based on the specific timing requirements.

The timing analysis is done based on the user constraints provided by the user. But the constraints written by Synplify are too many and this causes ISE to issue a waning about the number of constraints which would result in a long runtime. Hence the constraints from Synplify had to be consolidated into fewer common constraints. Especially the false path constraints.Grouping of constraints are achieved using the timegrp constraint explained further in this chapter.

The main constraints that need to be specified for an FPGA design are input paths, synchronous element to synchronous element constraint path, path specific exceptions, output paths and clocks.

Input timing constraints cover paths from the external pin of the FPGA device to the internal register that captures the data . This is achieved using the offset in constraint

[12]. The MSP430 prototype has all its clocks generated by the DCM, hence the main constraints of the design were at the interface of the dig-top hierarchy. So, the input data at the ports are not directly related to the input clock on the FPGA interface.

Register to register consraints cover the synchronous data path between the internal registers. This is achieved using the period constraint which:

- Defines timing requirements of the clock domain
- Analyze paths within a single clock domain
- Analyze paths between related clock domains

Path specific exceptions can be conveyed using the false path[10] and the multicycle path[11] constraints. Output paths cover the data path from registers inside FPGA to external pin of the FPGA.

The constraint is achieved by using the following main syntax for Xilinx

- tig: tig(timing ignore) constraint is used to false path all paths going through a specific net to be ignored for timing analyses and optimization.
- timegrp: it is a grouping constraint and is used extensively for consolidation of tig constraints from Synplify Pro.
- tnm and tnm_net are also grouping constraints ; these constraints are used identify elements that make up a group which can be used in a time_spec constraint.
- timespec is a timing constraint and is used in combination with the ts attribute to define the period of a timing group
- ts_identifier is used with the timespec for timing specification of an already defined timing group.
- the period constraint is a synthesis and timing constraint and is used to define internal clocks in a design and their period and optionally duty cycle

.

When a timing analysis is performed on an FPGA design the main checks carried out for each of the register to register paths defined in the constraints are

- Intrinsic clock to out delays of the registers
- Routing and logic delay

[10]are paths that are not valid like paths between asynchronous clocks or paths that are not timing critical

[11]paths between registers that take more than one clock cycle to stabilise

- Instrinsic setup and hold delay of the registers
- Clock skew between source and destination registers
- Component switching limits if the clocks are generated by clock modifying blocks.[12]

The following paragraphs describe the main register to register timing checks which verify setup and hold time compliance. Since one of the main issues in realizing this design was achieving timing closure[12] a basic explanation of the setup and hold time concepts are provided.

Setup time is the minimum time before the clock edge in which the data must be stable in order for the data to be propagated reliably. The figure 16 shows a basic figure of a setup violation. when the data is delayed more than the clock edge, the data can miss the clock edge and hence the data is not propagated over the destination register. The figure 18 shows that setup check is carried out over successive clock edges of the skewed clocks. Hence a setup violation can be fixed by lowering the frequency of operation.

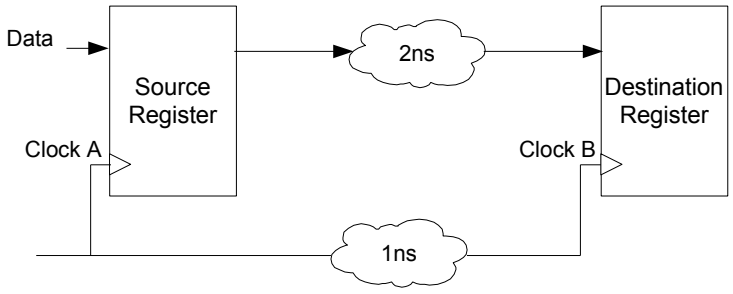


Figure 16: Setup violation

Hold time is the minimum time after the clock edge during which the data must be stable for reliable operation of the design. The figure 17 shows a possible hold violation scenario. The clock is delayed much more than the data. If the delay is large enough,

[12]implementing a design with no timing violations

when clka propagates the data on the source register, this new data can reach the destination register at the same clock edge from which it was launched from the source register thus overwriting the old data that had to be propagated through the destination register. The figure 18 shows that the hold time check is carried out on the same edge of the skewed clocks a and b, thus this means that the hold time checks are independent of the frequency of operation of the circuit. Hence, changing the frequency of operation would not fix hold time violations. So, care must be taken to avoid significant clock skews in the design.

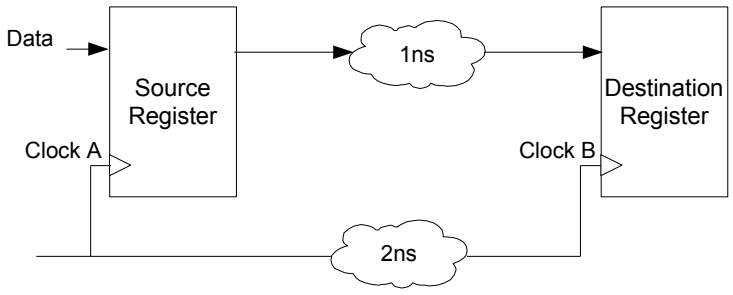


Figure 17: Hold violation

Finally figure 18 shows the clocks skewed clocks a and b. It also illustrates the clock edges used for setup and hold time checks.With this basis for setup and hold time analysis, a deeper look into how the timing analysis tools actually carry out these checks can be considered.

The slack for setup path is calculated using the following equation

$$slack = (requirement - (datapath - clockpathskew + uncertainty)) \quad (1)$$

Data path delay usually includes the following components.

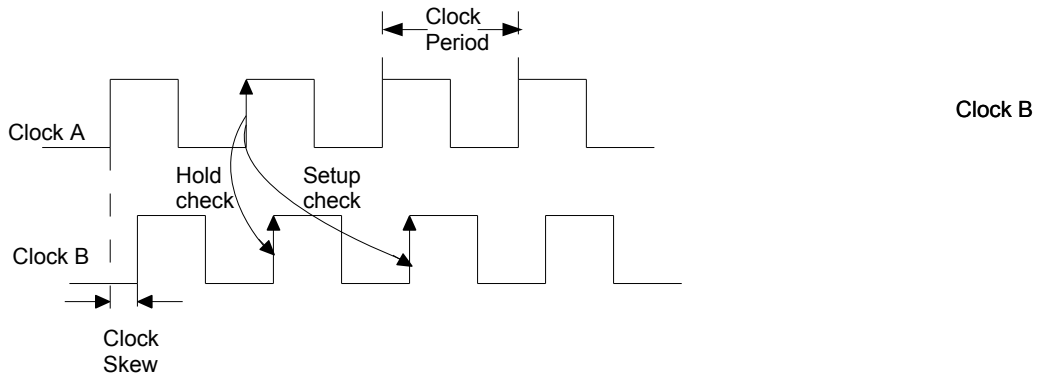


Figure 18: Clock edges for timing checks

- T_{cko} which is a sequential delay from flip flop clock clk to xq/yq output(clock to out)
- T_{ilo} which is combinatorial delays from the input of the LUT to the its output
- T_{as} setup time for address inputs with respect to clk.
- Along with these factors, the net delay introduced by routing is also accounted for.

Clock skew is calculated as the difference between clock path delay from the destination register to the source register. Clock path skew can be caused by one or both the clocks using local routing or one or both clocks being gated.

Clock uncertainity includes the following

$$\frac{\sqrt{(TSJ^2 + TIJ^2)} + DJ}{2} + PE \tag{2}$$

TSJ is the total system jitter
TIJ is the total input jitter
DJ is the discrete jitter
PE is the phase error

The slack for hold path is calculated similarly using the equation

$$slack = (requirement - (clockpathskew + uncertainity - datapath)) \tag{3}$$

From the equation it can be seen that as the data path delay increases the possibility for hold violation decreases. In general, due to the dominant routing delays present on an FPGA hold violations should not exist.

Constraint translation and consolidation: The constraints for the MSP430 design was taken from the ASIC design constraints. The task was to find the relevant constraints and apply them to the appropriate components on the FPGA. As mentioned earlier the constraints provided by Synplify were large close to 150 constraints and when this was passed through to Xilinx ise 11 there were issues with the number of constraints and hence they had to be consolidated. Consolidation was most essential while dealing with the false paths.

False path constraints are essential since it makes the task easier for the timing analysis tool to ignore these paths, in addition the map and par engine do not need to spend time and resources on trying to fix timing violations present on these paths. For comparison a pie chart of false paths and valid paths are shown in figure 19. From the figure it can seen that false paths are 9% of all the path analyzed by the tool, but looking at the magnitude of the number of paths the tool is dealing with makes this more significant. the timing analysis for the MSP430 analyses close to 55 billion paths. In addition the hold time violations in the false path groups have a magnitude as high as 6ns. Not accounting for all the false paths will result in additional runtime.

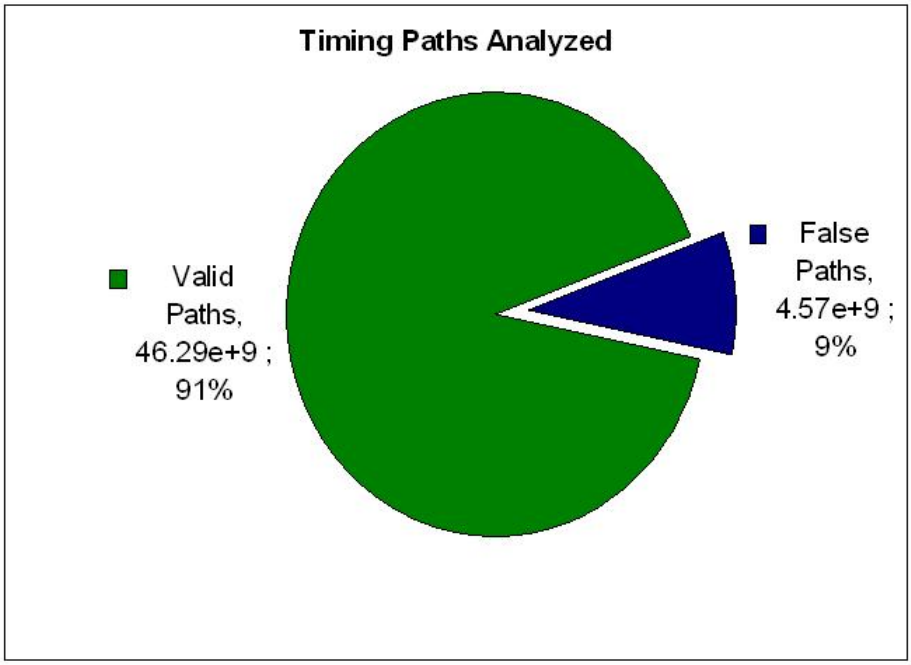


Figure 19: Timing path analyzed

9 Mapping and Place & route with PlanAhead

The PlanAhead tool is a design analysis,implementation and visualisation tool. PlanAhead can be used right from the synthesis to the post par timing analysis and bitgen. However, for our flow the PlanAhead is only introduced at the map and other post map processes. The main advantages of PlanAhead as discovered in the execution of the project are.

- Multiple design implementation strategies; this enables the designer to achieve timing closure quickly. The multiple stratergies can be run simultaneously on a multi-core processor further reducing the turnaround time
- Predictable and reproducable implementation with the use of pblocks and partitions
- Integrated environment with features for io planning, floor planning, static timing analysis, drc checks and wasso analysis that aid in detection of design problems at an early stage.
- Chipscope integration for debug capability.
- Time-ahead static timing analysis and TRCE post map or post par timing anal-

ysis.

- Visual analysis of the device which aids in the floorplanning procedure.

STA with PlanAhead and TRCE analysis: PlanAhead has two different timing engines TRCE and timeahead. The TRCE is used to perform a timing analysis at the end of map or par. The timeahead on the other hand does not model the routing but the timeahead is useful for floorplanning and timing estimation. The accuracy of the timeahead report improves with the newer devices, this early timing analysis can be used to guide the floorplanning. Estimation can be done with an estimate of interconnects or no interconnects. When no estimation is used the result is just based on logic delay. This may help in finding logic that has least amount of slack for routing delays. The estimated interconnect mode provides timing results that are accurate to within 20 to 30%[16]

The analysis from TRCE can be used to find the path of the logic and take corrective measures. All the logic elements in a path are selected when a corresponding timing analysis is chosen on the TRCE report. If the implementation result after par is present the path is also highlighted in the device view. In addition the schematic view can be used to view the corresponding path in terms of a register view as shown in the figures 21a and 21b.

Design preservation for predictable and reproducable implementation: Using design preservation enables the designer to reduce the number of implementation iterations in the final stages of the project. This is based on the principle of hierarchical design where the design is separated into a natural hierarchy of logic modules. While logic design with no hierarchy have the seeming advantage of better logic optimisation due to the absence of boundaries. Hierarchical design has the advantages of breaking the design into manageable logic blocks and also allows for preservation of results when one of the partitions achieve timing closure. Thus the design is broken into blocks called partitions. These partitions create an isolation from the rest of the design thus enabling the tools to export a successfully implemented partition and "cut and paste" the implemented result at the next rerun of the design implementation provided no changes have been made to the design within the partition.Thus preserving successful implementations of partitions and promoting them for successive runs has a considerable impact on the design runtimes. The MSP430 design also has a hierarchical design flow as shown in figure 20, but experiments with partitions were not carried out since it required guidelines like registering of outputs across partitions which were not considered in the MSP430 RTL.

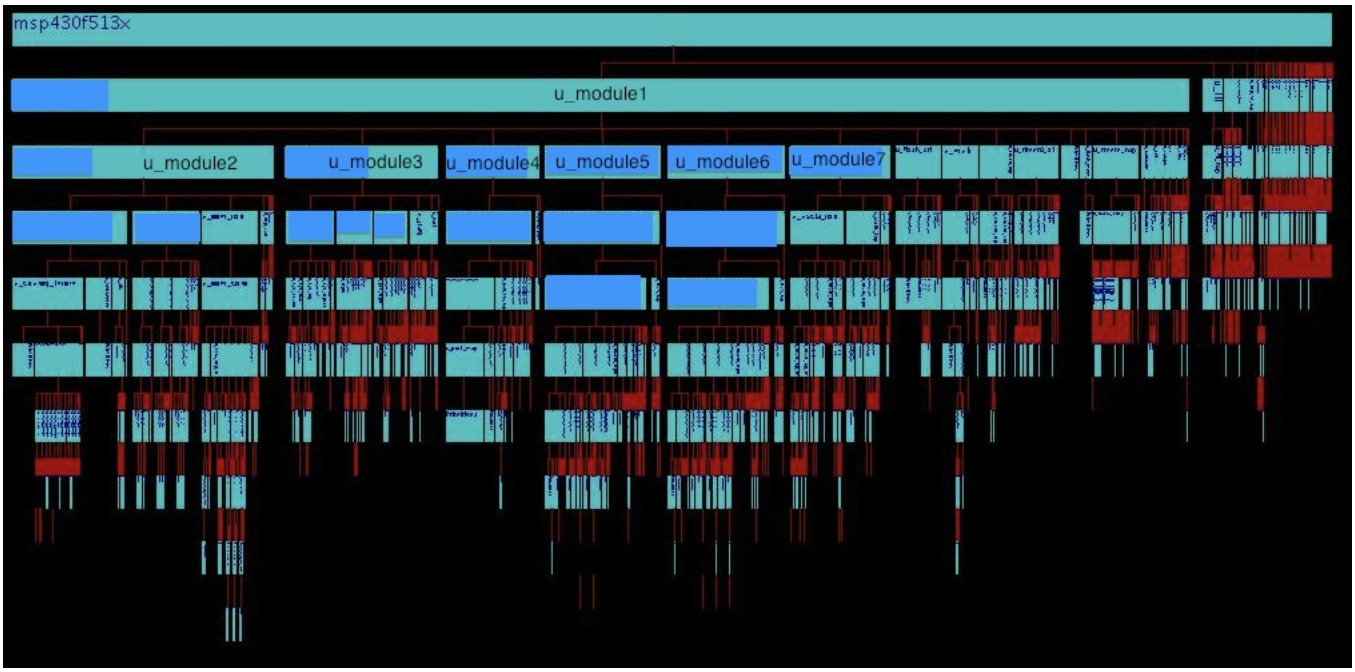


Figure 20: Hierarchical map of the design

IO planning: IO planner is the pin planning environment for PlanAhead. The IO pins can be locked by a drag and drop approach. They can also be locked by manually entering the site location. In addition to this IO planner is also capable of assigning iostandard, drive strength, slew type and pull type.

When using the IO planning environment for locking clock inputs on the IO pins, care has to be taken to ensure that the clock signal is assigned to the dedicated clock pins of the FPGA. Morever if using a single ended clock input instead of a differential input the p side of the differential IO pair has to be used for example use IO_L1P_xx instead of IO_L1N_xx for a single ended clock input. If these rules are not followed the placer issues the following error

"ERROR:Place:645 - A clock IOB clock component is not placed at an optimal clock IOB site. The clock IOB component zxlm is placed at site J4. The clock IO site can use the fast path between the IO and the Clock buffer/GCLK if the IOB is placed in the master Clock IOB Site. If this sub optimal condition is acceptable for this design, you may use the CLOCK DEDICATED ROUTE constraint in the .ucf file to demote this message to a WARNING and allow your design to continue. However, the use of this override is highly discouraged as it may lead to very poor timing results."

As the message explains design is not using a dedicated clock site to drive a global

clock buffer. These clock capable ios have a direct path to global routing which reduces routing delay. The error message can be demoted using the following constraint in the ucf file
NET zxlm CLOCK DEDICATED ROUTE = FALSE;

The data signals can be placed on any available user io, though generally for an efficient floorplanning it would be beneficial to have have a datapath that flows from the left to the right side of the FPGA.

Floor planning: Is used to guide the placement of modules during the implementation on the FPGA. Floorplanning might help reduce the routing delay component of the critical path. But if the critical path is dominated by logic delay, then synthesis constraints like source replication should be considered. Care must be taken while floorplanning as poor constraints would result in the degradation of the qor[13].

In addition floorplanning might also be used in a post par scenario when there are hold time violations that persist. In this case the total delay in the data path can be looked at. Normally for a FPGA the ratio between logic delay[14] to the delay introduced by the route delay[15] is 40:60 or 50:50. If there exists a hold violation where the logic delay contributes to the majority then such paths can be manually placed such that a higher percentage of routing delay is introduced. Surely, in such cases the impact of such modification to other parts of the logic have to be taken into account. This approach is a last resort in case the timing violations persist and must be taken up to achieve timing closure in the final steps.

DRC check: The design rule check carried out by the PlanAhead can be classified into logical and physical design rule check.

Logical DRC This is carried out in the following steps[9]

- Block check: this verifies that the terminal element in the hierarchy is an NGD primitive. It also checks that the user defined parameters are legal. any failure results in an error.
- Net check: this check determines the number of output pins (drivers) and input pins(loads) on each signal in the device. If a signal does not have at least one

[13]Quality of Results
[14]Delay introduced by logic operation from blocks such as LUT,CLB,memory blocks
[15]Delay introduced by the interconnect in terms of routing and fanout

driver and one load a warning is generated. If a signal has multiple drivers an error is generated.

- Pad check: this verifies that each signal connected to the pad obeys the following rules.
  - If the pad is input type it can only be connected to the following buffers, clock buffers, pull up, pull down, keeper and bscan.
  - If the pad is output the signal attached to it can only be connected to a single primitive of the following buffer, 3-state, bscan, pullup, pulldown, keeper.
  - If the pad is bidirectional it must obey the rules for input and output pads, the signal attached to it must be declared as bidirectional.
  - If a signal is connected to the pad has a connection to a top level pin, the top level symbol must have the same type as the pad pin.
  - A signal connected to multiple pads results in a warning.
- Clock buffer check: Verifies that the output of clock buffer connects to inverter,flip flops,latch primitive clock inputs or other clock buffer inputs. non compliance results in a warning.
- Name check: verifies the uniqueness of the NGD primitives, the rules for name check are
  - Pin names must be unique, violation leads to error.
  - Instance name must be unique in a branch of hierarchy, violation results in warning
    Signal names must be unique in a hierarchy branch, violation results in a warning.
  - Global signals must be unique, violation results in warning.
- Primitive pin check: Verifies that certain pins on the primitives are connected to signals. Such as select line on a mux.

Physical drc The physical drc checks for physical errors and logic errors in the design. Physical drc is run after map, PAR and before bitgen. The steps involved in a physical DRC check are:

- Net check: similar to net check of logical drc.
- Block check: Checks placed and unplaced components and reports problems with pin connection, logic or configuration.
- Chip check: Performs checks at the chip level for nets and components such as placement rules.

Runtime and PAR effort: PlanAhead has a preset of design goals and strategies[16] that enable the user to choose the right tradeoffs for implementation of the synthesized EDIF. However the user can also create a custom strategy to suit his needs.

For the implementation of the MSP430F5172 , the design strategy used high effort and a normal extra effort for both map and place and route. The difference between the effort levels of the router are as follows:

- Standard: Gives a fast run time with lowest routing effort. Appropriate for a less complex design.
- Medium : Gives a slower run time with some routing optimization.
- High: Gives the best routing results, but will incur the longest run time. Appropriate for a more complex design.

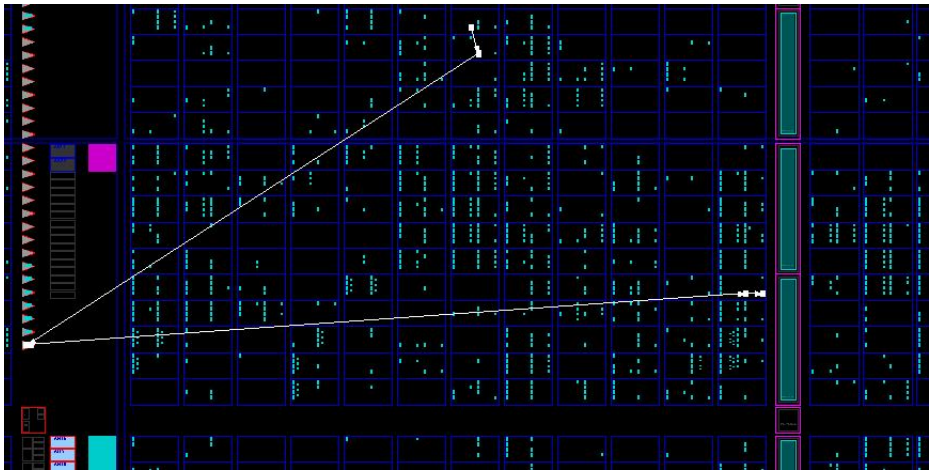
Similarly the extra effort level for the router is as follows, the extra effort puts in additional runtime to meet the specified design constraints.

- None: No extra effort level is applied.
- Normal: Runs until timing constraints are met unless they are found to be impossible to meet. This option focuses on meeting timing constraints.
- Continue on Impossible: Continues working to improve timing until no more progress is made, even if timing constraints are impossible. This option focuses on getting close to meeting timing constraints.[10]

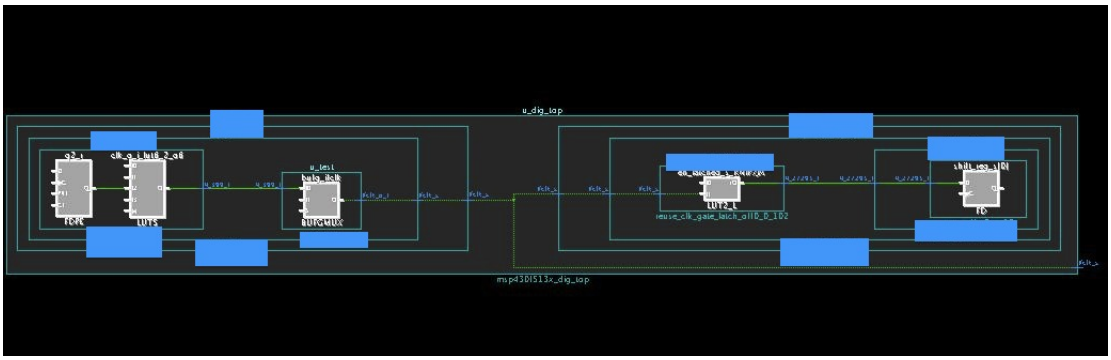
The reason for using such a "severe" effort is due to the synthesis issues of Synplify Pro discussed on page 34. The implementation tool was able to produce a timing clean netlist even without clock gating conversion enabled.

Result visualisation: One of the advantages of using PlanAhead instead of Xilinx ISE for the map and PAR processes was the visual interface and overview it provided during the execution of the processes. To give an example post PAR PlanAhead generates a timing report which details the paths of the user defined timing groups and gives us detailed information of paths analyzed and timing violations if any. The user can choose to view these paths as a schematic on the FPGA fabric. The figure 21 gives us such an example.

[16]Design Goal and strategies contain predetermined set of process properties that have been selected to achieve a particular goal like Power optimization,Area Reduction,Minimum runtime,Timing Performance and Balanced.Design Goals and Strategy enable the user to guide the implementation tool to achieve optimal results based on the user choice listed in the previous line. Design strategy are saved in a datafile with .xds extension.Custom design strategies can be specified by the user[17]



(a) FPGA fabric view



(b) schematic view

Figure 21: PlanAhead schematic views

Figure 21a gives us the representation of the placement of logic on the FPGA, this view helps us visualise the routing resources involved in the datapath under observation. Whereas figure 21b gives us the schematic of the logic actually implemented in terms of the FPGA primitives.

10 Debugging with FPGA editor

The FPGA editor is a GUI for editing and viewing the NCD file[17]. The FPGA editor is a power tool and gives access to the basic logic and routing fabric of the FPGA. Undoubtedly such a power tool has extensive features and capabilities since it taps into the very last levels of FPGA implementation, this also makes it more complex also taking into account that it is not documented extensively. The most useful functions for this project have been. The same functions are discussed in this section and relevant results are also presented.

- Debugging internal FPGA signals with probes
- Manual routing and unrouting data and clock lines.
- Deleting or unplacing components.
- Changing configuration of LUTs, pull states of pads as well as level sensitivity of logic.
- Usage of scripts that enabling automation of the design
- Other functions include
  - Place and route critical components before running the automatic par
  - Finish PAR if the routing program does not completely route your design
  - View and change the nets connected to the capture units of an ILA core in your design
  - Use the ILA command to write a .cdc file.

Possibility to change attributes of FPGA resources: This possibility provides extended flexibility to the user. The FPGA editor not only allows for the modification of the attributes of the components but also creation of new components. This is done in the block window[18] as shown in the figure 22.

[17]Native circuit description is an output after map and after place and route.It contains logic of the design mapped to components such as CLB and iobs.

[18]this window is displayed when you double click on a logic block

LUT and flip flops have attributes associated with it in the form of check boxes as shown in 22c. So in order to change the attributes, the user just needs to change the check boxes in the edit mode. Some of the attributes associated with the CLB are as follows

- The logic element can be a ff/latch
- Initial values may be '0' or '1'
- Default value after set and reset may be '1' or '0'
- Each CLB has muxes to route the output, clicking on the triangle enables he user to select from the inputs.
- It is also possible write a new LUT equation for the output. A typical LUT equation looks like $A5 * (A2 * (\sim A1 + A4)))$. The representation for the logic symbols in the FPGA editor are as described in table 12

Table 12: FPGA editor logic operation

symbol	logical gate
$\sim$	not
$+$	or
$*$	and
@	xor

Selective attribute modification is explained in the section taking the example of a DCM.

Selective manual unrouting: To manually unroute a wire from a component or net pin, just select and click delete or select unroute. If one needs to unroute all the wires connected a component, unplacing the component achieves this objective as well as removing the component from the site.

This approach was particularly useful when increasing the frequency of one of the internally generated clocks. As seen in digital clock manager paragraph on page 11. The input clock frequency is limited when the DCM operates in max speed high frequency mode, in such cases the DLL mode output of CLK0 cannot be used and the feedback mode is also not possible. In such cases the whole flow from synthesis to par and route can be run taking about 3 hours to complete or the same objective can be achieved using the FPGA editor in a couple of minutes. To achieve this the following actions need to be carried out

- The feedback attribute of the DCM has to be changed to no feedback mode
- The routing from the CLK0 output to the BUFG to the CLKFB has to be unrouted
- The multiplication factor has to be changed for the required frequency, care must be taken since there are no component switching limits check that is carried out.

These steps are illustrated with the help of the figures 22a,22b and 22c.

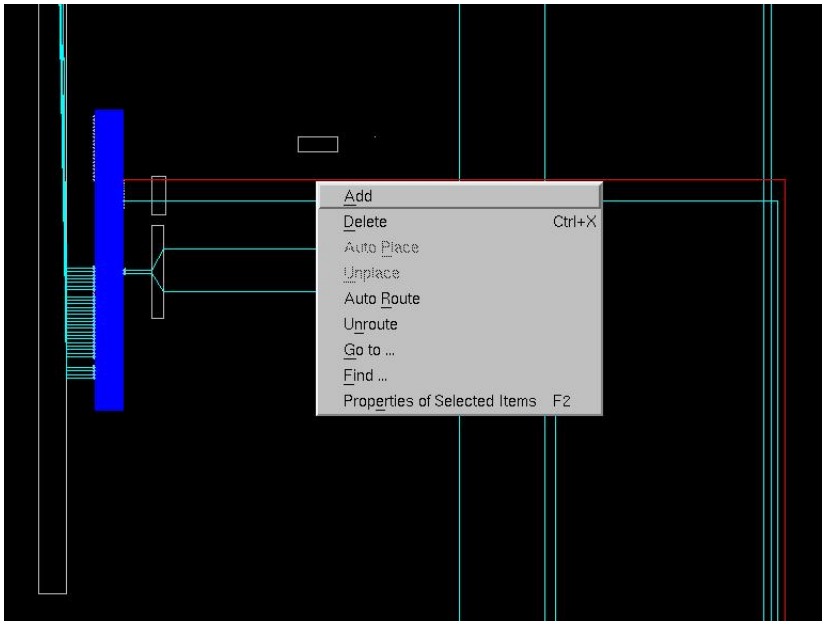
Probes for low level debugging: Probes are extremely useful for debugging internal signals in the design. Probes can access any signal in the FPGA except signals that are internally routed in a slice. The selected signal is routed to an unused pin on the FPGA. The probes give access to real time behavior of the signals in their actual operating environment.

Using the probes is a good workaround for debugging the FPGA without having to go through the learning curve involved in chipscope. Unlike chipscope it does not require additional hardware. however the disadvantages of this method of debugging is that it is hard to access bus signals and multiple triggers are also hard to access. The manual effort involved in creating probes for buses which are 32 bits wide or of similar magnitude can be circumvented to some extent by the use of scripts which are discussed in the scripts paragraph on page 54.

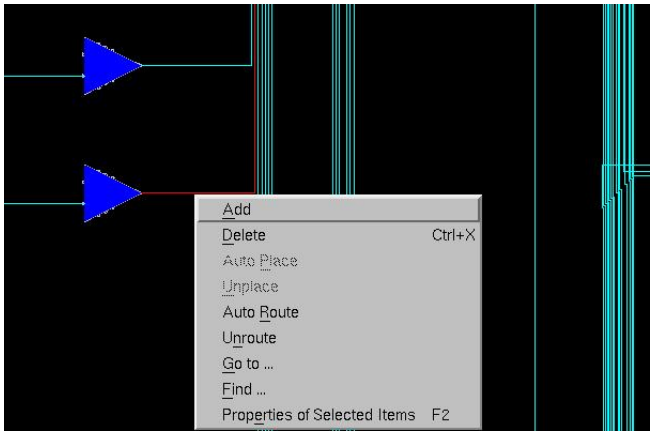
Probes are very useful for debugging the design. But the inconvenience in finding the necessary net for debugging is that the nets are renamed during the whole process of synthesis,mapping and PAR. It is very common to find logic optimised and nets renamed to names like n_551_s, n_2145 and so on. By using the synthesis directive of syn_keep[19] we can preserve the net and the net name for easy debugging. But the use of this synthesis directive would require resynthesis, re map and re PAR. Thus, defeating the quick turnaround advantage of using probes. Moreover, the logic implemented in a LUT is displayed as $A5 + (A2 * (A1 + A4)))$ which is not a very intuitive way to visualise a logic equation. The RTL and technology view from Synplify Pro can help overcome these obstacles. As shown in figure 15b the implemented logic on LUTs is still depicted as logic gates thus enabling the designer to go through the datapath in a very intuitive manner. In addition the crossprobing [20] functionality of the Synplify Pro enables us to view the corresponding logic generating abstract nets like n_3145 and also view their RTL code. Thus, the interlink between the different tools are essential

[19]keeps the specified intact during optimization and synthesis

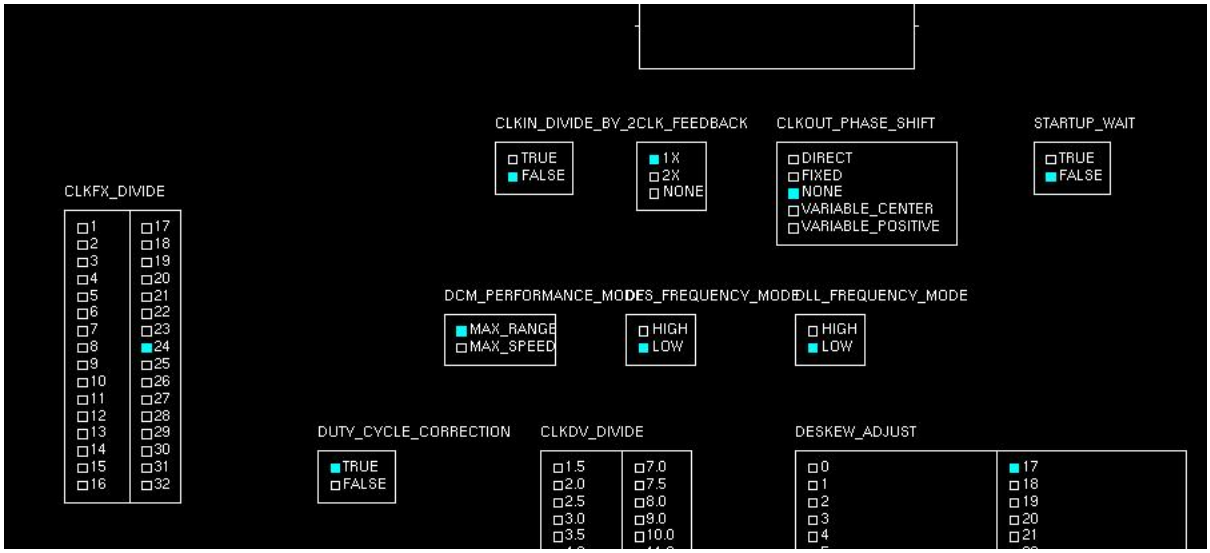
[20]process of selecting an object in view and having the object or corresponding logic automatically highlighted in other views like RTL,technology view or RTL code.



(a) Unroute clk0 output



(b) Unroute bufg output and delete bufg component



(c) Change attributes of DCM

Figure 22: Manual editing with FPGA editor

throughout the debugging process involved in this flow.

Net delay investigation: This investigation is useful to find the actual delay on the net after PAR. This is the information used to plot the histogram of the ifclk in the figure 27. The net delay can also be used to autoroute the design based on the users constraint for individual nets.

For the example where the ifclk uses local routing, the design might need a more predictable routing to achieve timing closure. In such cases the FPGA editor can be used to minimise the local routing deviation. By choosing the appropriate net(in this case ifclk) unroute the whole network, and later from the properties window set the maximum delay of the net to 2ns. Then use the autoroute functionality let the FPGA editor find a more suitable routing to satisfy this constraint. The result is again shown in the histogram 27 which displays a much smaller deviation. But it also need to be considered that the FPGA utilisation for this design was just 24% hence there might be more routing resources that are available to satisfy the constraint on the net. Also looking at figure **??** it can be seen this constraint of 2ns is indeed a tight constraint and the tool tries to make use of as many long lines[21] as possible to achieve this constraint. In case of design with more congestion and device utilisation a significant improvement might not be achieved. The result from this experiment is shown in figure 27.

Scripts: Scripts are a powerful feature of the FPGA editor that enable the user to replicate his actions automatically. An example script is given in the appendix B. The script feature of the FPGA editor is very easy and intuitive to use. The tool records the actions of the user and outputs these to a script file, which can later be modified by the user to suit his needs. This approach does not require a steep learning curve or prior knowledge of scripting but does provide significant reduction in effort and time involved in repetitive actions.

Routing resources: Since there has been considerable discussion about routing resources in this report, this paragraph attempts to provide an overview of the routing resources of the FPGA along with some snapshots of the resources on a Virtex 5 FPGA in figure 23. The documentation regarding FPGA routing resources are very sparse, the information here is taken from the FPGA editor user guide[13].

- Local lines: Usually span across multiple CLBs; typically they go between switch boxes. Local lines do not directly connect to site pins, such as direct connects,

[21]for more information refer to the paragraph on routing resources

and they do not span across the entire length of the device, such as long lines. The figure 23c shows local line routing in grey.

- Long Lines : A long line connects to a primary global net or to any secondary global net. Each CLB has four dedicated long lines. Long lines carry signals across the entire length or width of the chip with minimal delay and negligible skew. The figure 23b shows a long line in purple.
- Pin wires : Are directly tied to the pin of a site such as CLBS and these are illustrated in green in the figure 23c.

Timing path crossprobing in FPGA editor: Just like the Synplify Pro and PlanAhead schematic representation of logic, the FPGA editor also enables the user to view the implemented design on the FPGA. But the difference here being that the view on the FPGA editor gives the actual view of the FPGA fabric complete with switch boxes (the bigger white blank rectangles)[22], routing resources (in red) and CLBs (in blue) as they occur on the FPGA as shown in figure 21a. Just like the PlanAhead allows the crossprobing between timing report and schematic, the FPGA editor provides a similar functionality but this view as shown in figure 24 has the addition of the details of routing resources that is absent in the PlanAhead view as in figure 21a.

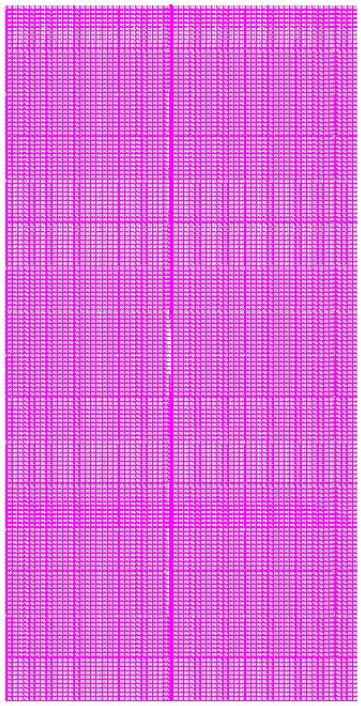
Resource utilisation: For completeness, a graph of the resource utilisation is provided in the figure 25 which shows the LUT utilisation is about 24%.

11 Other issues faced

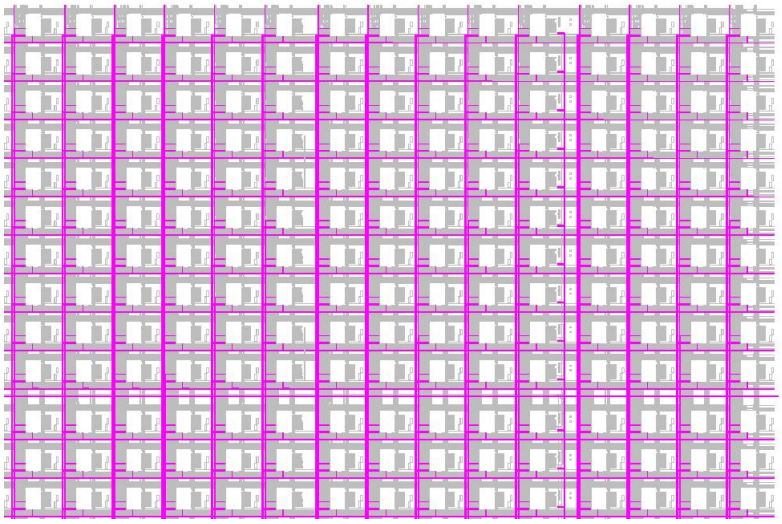
Apart from all the issues and solutions discussed in the previous pages this section provides an overview of the other issues faced during the execution of the project.

Buffering for high fanout clocks like ifclk: The ifclk with a fanout of about 3300 is the output of a mux. Such high fanout clock nets must use a global routing resource. For this particular example instead of using a BUFG after the mux, the BUFGMUX primitive can be used. Such clock selection is quite common in the MSP430 RTL. But due to the limited global resources available not all the clock muxes can be replaced by BUFG primitives. On the other hand if the clock can be restricted to 3 adjacent

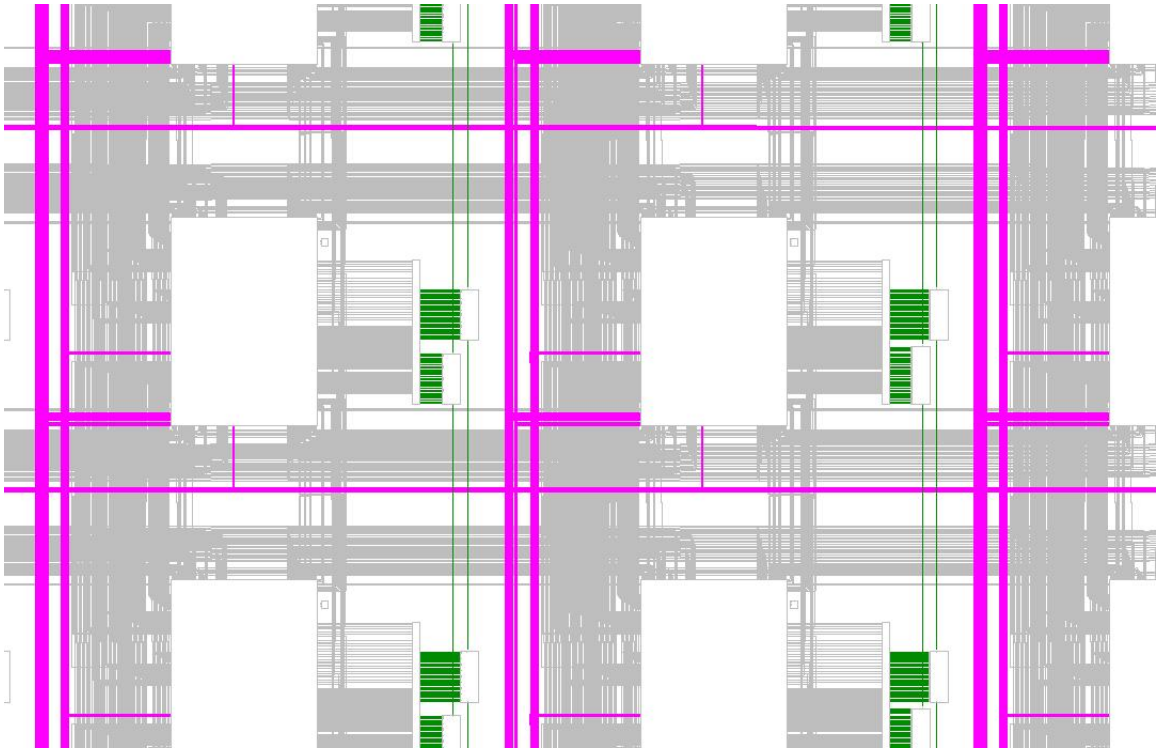
[22]collection of transistors located between CLB blocks that enables the connection of two interconnect lines



(a) Device view of Virtex5 for routing resources.

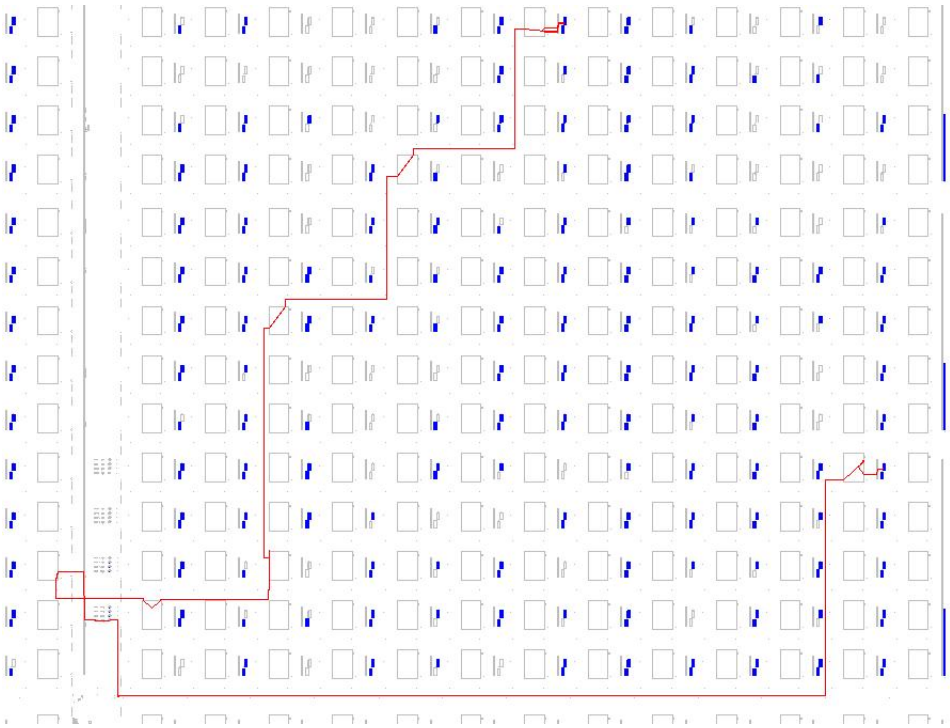


(b) Zoom view of long line routing resource

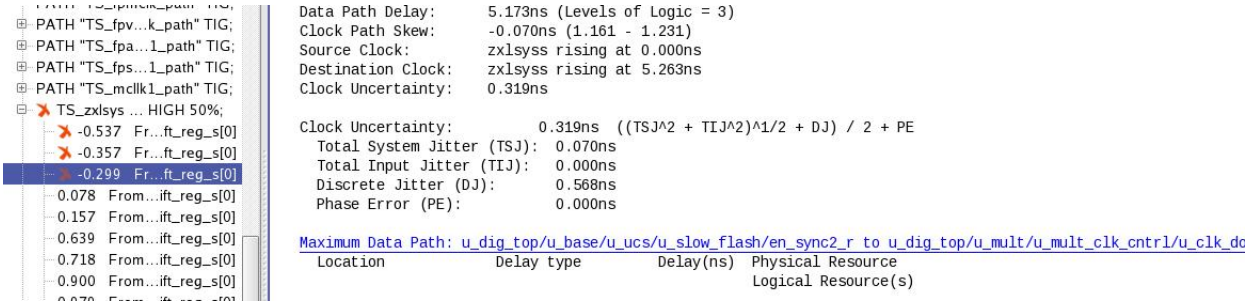


(c) Zoom view of local lines and pin wires

Figure 23: Virtex5 routing resources.



(a) FPGA fabric view



(b) Corresponding timing report

Figure 24: FPGA editor schematic views

clock regions the output of these clock mux can be routed through a regional clock buffer(BUFR). The following figure shows the routing of the ifclk on the FPGA.

The figure **??** does suggest a sense of uniformity when using the global resources over the local resources. This is confirmed by calculating the average and standard deviation values for the net delays using each of the above types of resources. The standard deviation for the global, local and constrained local routing are 0.167, 0.919 and 0.606. Hence even running a constrained local routing does give us a better standard deviation compared to unconstrained local routing, but this is still not as good as using global routing resources.The following histogram in figure 27 the variation of the route delay of the ifclk.

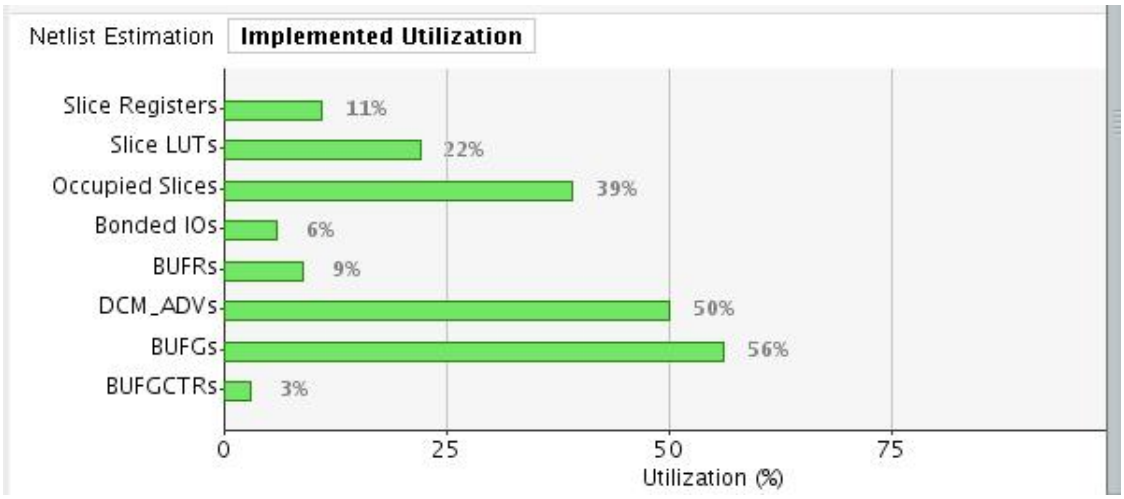


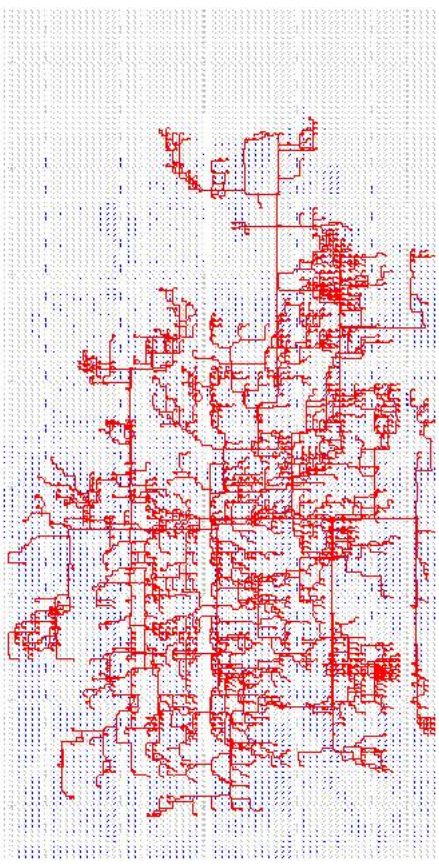
Figure 25: Resource Utilisation

From the histogram it can be seen that the use of global is essential to have uniform clock net delay. For this design there were about 30-40 setup errors when the ifclk used local routing. But another reason for opting for global routing for nets with high fanout is to avoid congestion. By using the global routing resources the local routing resources are freed up for less critical nets.

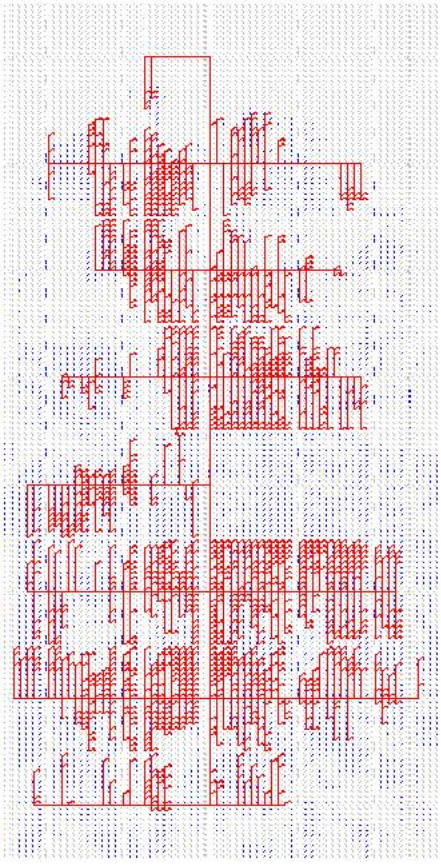
PUC and POR not being deactivated: After going through the flow and obtaining a timing clean netlist and the bitgen flow was cleared. The device had to be checked for JTAG access. The following paragraphs describe the issues faced while getting the JTAG access to work.

The first issue was getting the device out of reset. Probe feature of the FPGA editor discussed on page 52 was most useful at this stage of the project. This enabled us to access the signals quickly thus saving us a lot of time.

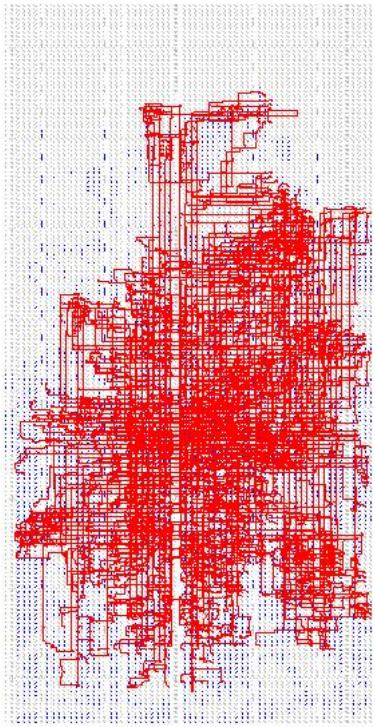
The first issue that was identified as the cause was that the TEST pin is supposed to be low else the activity on the RESET pin is not reflected on the POR. Additionally it can also be observed from the figure 28 that when the TEST pin is high the previous state of the POR is latched. A missing latch in the top level description prevented this behavior. After this issue was rectified the POR behavior was as desired.



(a) ifclk local routing



(b) ifclk global routing



(c) ifclk with autoroute from FPGA editor

Figure 26: ifclk routing resources.

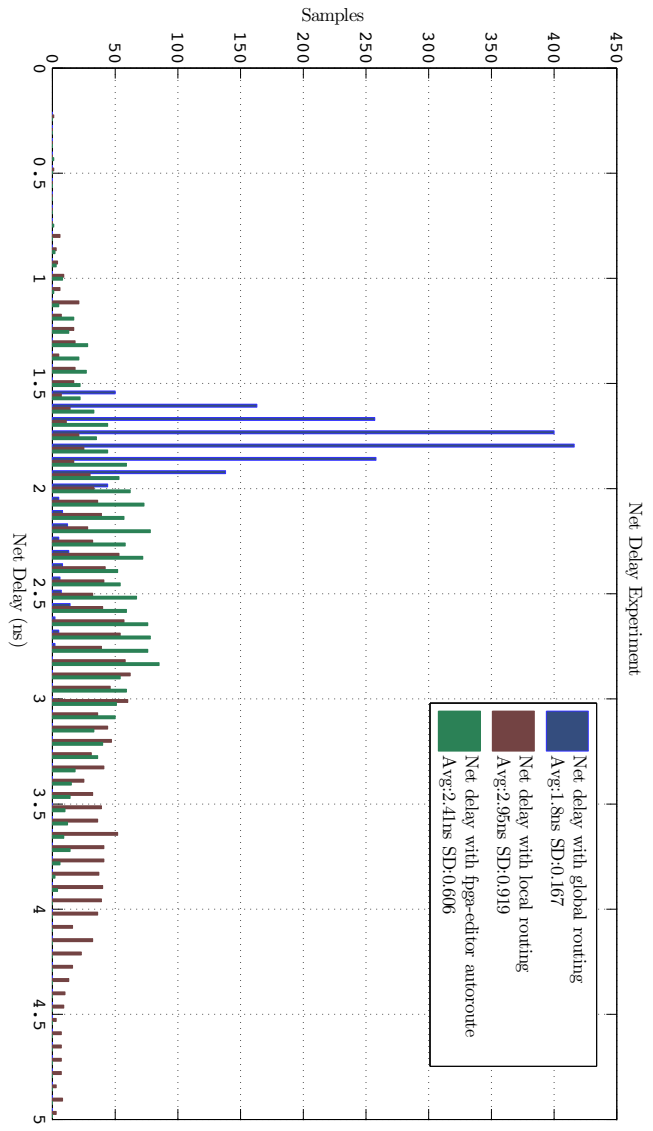


Figure 27: ifclk net delay with global and local routing

Locked IOs prevented JTAG access The above solution provided the right functionality for the reset signal, but JTAG access was not yet established. On further debugging using probes it was discovered that the JTAG tck signal was not getting past the port logic. The root cause for this issue was an initial value of '1' on a register

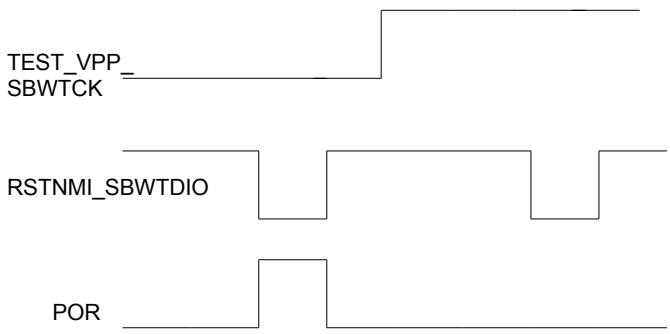


Figure 28: Latching of POR with TEST signal.

that caused the device low power mode signal to be activated which in turn locked the IO ports and did not propagate the changes on the pins. After changing the initial values JTAG access was finally achieved.

Memory access As discussed in the section 3 the behavior of the MSP430 flash memory is:

- Flash initial value is all ones.
- Only a one can be programmed to zero and not the other way around

So to check if the previous memory contents were indeed all ones a simple AND operation is performed between the read data and the data being written. Since the BRAM of the Virtex are synchronous memories, incorrect data was being written when the same clock edge was used for both read and write operation. Thus the read had to performed on the positive edge of the read clock and write on the negative edge of the write clock. This problem was overcome by manually changing the edge sensitivity of the BRAMs on the FPGA editor as discussed on page 51.

12 Results with functional test of the FPGA prototype

In order to test the functionality of the MSP430 prototype the FPGA was put to some basic tests. The tests and their results are discussed in the following sections.

Maximum frequency of one of the peripheral: This test was done to test the maximum frequency of operation for one of the modules of the MSP430. This particular peripheral was selected because for ZeBu the maximum frequency of operation was 32 MHz. Whereas after tests it was seen that for the FPGA prototype the maximum frequency of operation was 125 Mhz. The method was verifying this was the phase difference of two output waveforms as shown in figure 29.

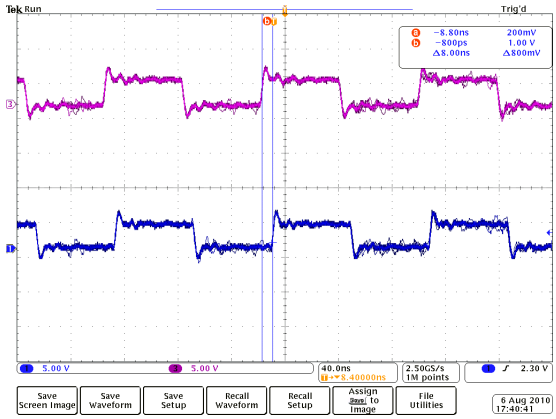


Figure 29: Maximum frequency of operation

The phase difference between the two waveforms correspond to the minimum period the module can operate at. For our prototype this is 8 ns or about 124 Mhz. This limitation was just posed by the maximum frequency of operation of the DCM under low frequency and maximum range of operation as described in the DCM section on page 11 and in the figure 6. However, the frequency can be increased by cascading DCMs. This was not tested.

Low power mode test: The test carried out for the low power modes was just to test the emulation of low power modes. That is mainly to verify that the clocks are switching off. As described in section 3 and in table 2. All the low power modes were tested for disabling of the clocks. The colour legend for the figures 30 and 31 are as shown in the table.

The code template used is shown in appendix D. The device after configuration is put into one of the low power modes and the device is woken out of LPM by an interrupt on the P1.4 [low to high transition]. Every interrupt toggles the output on pin P1.1 which is the shown as signal in purple in the figures 30.

The first example shows the configuration for low power mode 1 as seen in table 2 in LPM1 the SMCLK can be optionally active and this is the configuration chosen. When

Table 13: Colour legend

Colour	Clock
Light Blue	MCLK
Dark Blue	SMCLK
Green	ACLK
Purple	Port 1.4 interrupt

the device is in LPM1 the MCLK is disabled, SMCLK is always active and ACLK is active by default.

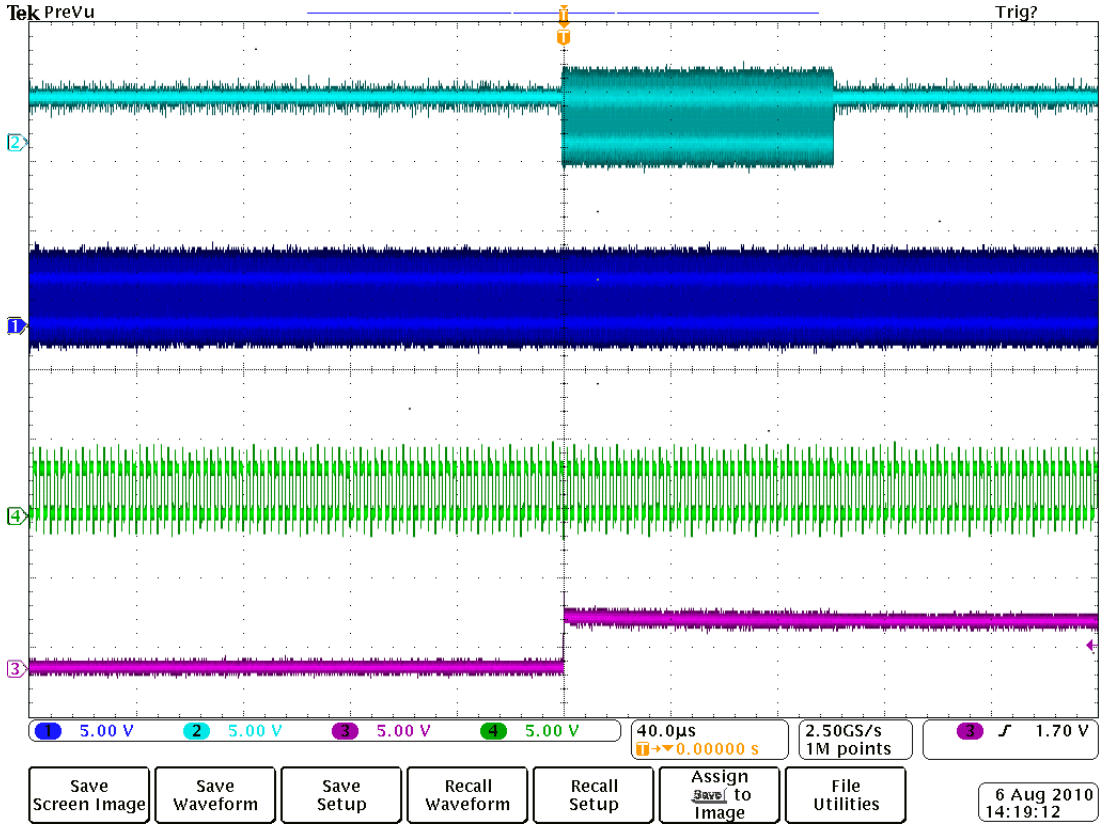


Figure 30: Low power mode 1

The next example the device is put in low power mode 4. In LPM4 all the clocks are disabled as shown in the figure 31. It can also be noted that every time p1.1 toggles it means there has been an interrupt on the p1.4 to wake the device out of low power mode. For more information on digital IO of the MSP430 refer to the paragraph on the same on page 5.

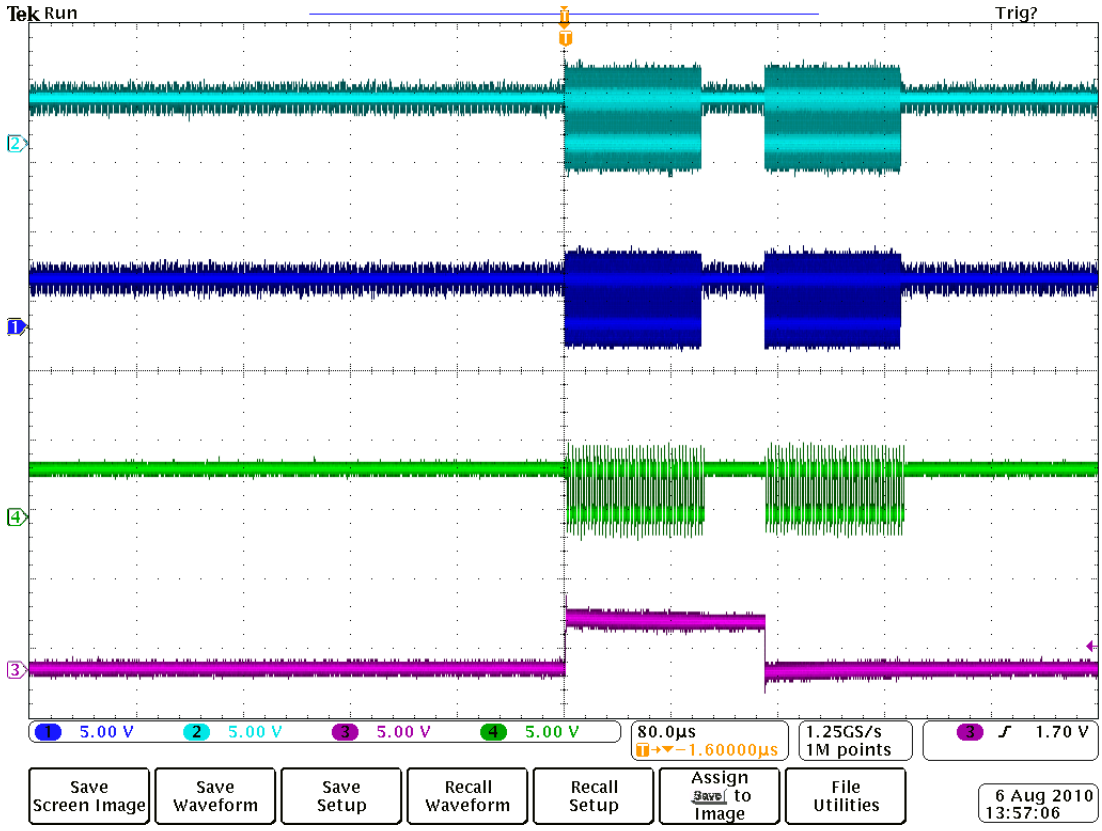


Figure 31: Low power mode 4

Low power modes of the MSP430 are based on the fact that reducing clock activity reduces dynamic power consumption. Since total power consumption of a circuit is a sum of static[23], dynamic[24] and leakage power[25] consumption; reducing dynamic power consumption brings down the total power consumption of the MSP430.

The clocks MCLK and SMCLK were observed on pins p3.4 and p3.2 respectively. There are also dedicated pins for the MCLK,SMCLK and ACLK on the pj port of the MSP430F5172 but since the pj port was used for JTAG access, the ACLK was observed using the probe of the FPGA editor, where the ACLK was routed to an unused io port as discussed on page 52.

The tests that were carried out not only tested the low power modes but also the port functionality of the MSP430F5172 prototype. In addition the tests also confirmed the proper functioning of the emulated flash on the FPGA BRAMs.

[23]due to subthreshold currents, even though the transitors are"off"

[24]due to charging and discharging of capacitors driven by gates in the circuit

[25]due to direct path between Vcc and ground,ex:when both nmos and pmos are conducting in a cmos circuit.

These tests were carried out using the emulator function of the IAR embedded software[26]. JTAG access was used to communicate with the device.

13 Conclusion

From the tests carried out in the section 12 it can be seen that the FPGA prototype is functionally verified. However, more extensive tests have to be carried out to establish the limitations of such a prototype.

The work carried out during the duration of the thesis indicate that the modern ASIC designs are growing more and more complex with concepts like power and clock gating being used extensively. It has also shown that FPGA implementation tools still need to keep up with such growing design techniques, but however quickly implementation tools evolve to adapt to these growing design techniques there are still possibilities of tool bugs as discussed on page 34. On the other hand, results have shown that newer tools do have algorithms that enable timing closure even for FPGA designs where gated clock conversion has not been enabled, though at the cost of extra runtime. This also questions the "traditional" view that gated clock conversion in FPGA prototypes of ASIC is one of the most important steps.

Future work in this project would include memory expansion for the FPGA in terms of interfacing with an external memory to accommodate the higher memory requirements of future MSP430 families. A trial with partitions that enable runtime reduction, however it has to considered what level of effort has to be put in by the designer to achieve a significant runtime reduction. Gated clock conversion could be tried even for multiplexed clocks with the principles introduced in page 32.

Appendix A FPGA RTL code samples

A.1 Package declaration for global signals

– Title : Package declaration for the MSP430F5172 FPGA prototype
– Description: Package declaration for delivering global signals mainly clock signals

[26]Development tool for embedded systems.

–Date :2010 july 9

```
library IEEE;
use IEEE.std_logic_1164.all;

package zxlsys_glb is
signal zxl_clkh,zxl_clkm,zxl_clkidel : std_logic;
end package zxlsys_glb;


library IEEE;
use IEEE.std_logic_1164.all;

package clkinj_glb is
signal refo_clk,vlo_clk,modosc_clk,dco_clk,pssclk_clk,lfxt_clk,pgen_clk: std_logic;
end package clkinj_glb;
```

A.2 Clock injection modules using global signals

– Title : Clock injection module for the MSP430F5172 FPGA prototype
– Description: Entity declaration to instantiate the global clock signals without adding ports on the modules.
–Date :2010 july 9

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.clkinj_glb.all;

entity clkinj is
port (refo,vlo,modosc,dco,pssclk,lfxt,pgen1:out std_logic
);
end entity;

architecture arch_clkinj of clkinj is
begin
refo <=refo_clk;
```

```
vlo <=vlo_clk;
modosc <=modosc_clk;
dco<=dco_clk;
pssclk<=pssclk_clk;
lfxt<=lfxt_clk;
pgen<=pgen_clk;

end arch_clkinj;
```

– Title : Clock injection module MSP430F5172 FPGA prototype
– Description: Entity declaration to instantiate the global clock signals without adding ports on the modules.
–Date :2010 july 9

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.zxlsys_glb.all;
library unisim;
use unisim.vcomponents.all;

entity zxlsys is
port (
clkh :out std_logic;
clkm:out std_logic;
clkidel : out std_logic
);
end entity;

architecture arch_zxlsys of zxlsys is

begin

clkh <= zxl_clkh;
clkm <= zxl_clkm;
clkidel <= zxl_clkidel;

end arch_zxlsys;
```

A.3 Clock generation module

```
----------------------------------------------------------------------
– Title : Clock generator for the MSP430F5172 FPGA prototype
– Description: Generates clocks of required frequencies based on a 24mHz clock input.
DCM components used for clock manipulation.
–Date :2010 july 9
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.zxlsys_glb.all;
use work.clkinj_glb.all;
library unisim;
use unisim.vcomponents.all;
library Synplify;


entity clock_block is
port (
CLK_PAD : in std_logic;
CLKFX_OUT_pgen : OUT STD_LOGIC;
CLKFX_OUT_dco : OUT STD_LOGIC;
CLKFX_OUT_lfxt : OUT STD_LOGIC;
CLKFX_OUT_vlo : OUT STD_LOGIC;
CLKFX_OUT_modclk : OUT STD_LOGIC;
CLKFX_OUT_ref : OUT STD_LOGIC;
CLKFX_OUT_pssclk : OUT STD_LOGIC;
CLKFX_OUT_zxlsys : OUT STD_LOGIC;
CLKOUT_zxlmh: out std_logic;
config_rst1: out std_logic;
clk_out :out std_logic


);

end clock_block;

architecture STRUCT of clock_block is

signal CLK,CLK90_tmp,CLK_tmp90, CLK_int, CLK_USB,clk0_USB,CLKFX_OUT_tmp,CLK_tmp
: std_logic;
signal CLKFX_OUT_pgentmp,CLKFX_OUT_dcotmp,CLKFX_OUT_lfxttmp,CLKFX_OUT_vlotmp,
CLKFX_OUT_modclktmp,CLKFX_OUT_reftmp,CLKFX_OUT_pssclktmp,CLKFX_OUT_zxlsystmp
```

```
: std_logic;

SIGNAL CLK_PGEN,CLK0_PGEN,CLK_DCO,CLK0_DCO,–CLK_LFXT,CLK0_LFXT,CLK_VLO,CLK
CLK_MODCLK,CLK0_MODCLK,CLK_PSSCLK,CLK0_PSSCLK,CLK_ZXLSYS,CLK0_ZXLSYS
: STD_LOGIC;

SIGNAL CLKDIV_OUT_LFXT,CLKDIV_OUT_REF,CLKDIV_OUT_tmp,CLKDIV_OUT_VLO,CLKFX
CLK0_DCM2,CLK_DCM2,CLKFX_OUT_DCM2TMP,clkfx_out_DCM11,LOCKED_PSSCLK,LOCKED_

signal config_rst : std_logic := '1';
signal config_rst_bor : std_logic := '1';
signal rst_DLL,rst_int1_tmp,CLKFX_OUT_ZXLSYS1,CLKFX_OUT_zxlsys2,clkfx_out_zxlsys3,zxl_lock
: STD_LOGIC ;
SIGNAL CLKFX_OUT_VLO1,CLKFX_OUT_DCO1 : STD_LOGIC;



component BUF
port (
I : in std_logic;
O : out std_logic);
end component;

component BUFG
port (
I : in std_logic; O : out std_logic);
end component;



component DCM_BASE is
generic (CLK_FEEDBACK : STRING;
CLKDV_DIVIDE : REAL ;
CLKFX_DIVIDE : INTEGER;
CLKFX_MULTIPLY : INTEGER;
DLL_FREQUENCY_MODE:STRING;
DFS_FREQUENCY_MODE:STRING;
CLKIN_PERIOD: REAL;
DCM_PERFORMANCE_MODE : STRING);



port (
```

```
CLKFB : in std_logic;
CLKIN : in std_logic;
RST : in std_logic;

CLK0 : out std_logic;
CLK90 : out std_logic;
CLK180 : out std_logic;
CLK270 : out std_logic;
CLK2X : out std_logic;
CLK2X180 : out std_logic;
CLKDV : out std_logic;
CLKFX : out std_logic;
CLKFX180 : out std_logic;
LOCKED : out std_logic );



end component;



signal logic_0 : std_logic;
attribute syn_noprune : boolean;
attribute syn_black_box : boolean;
attribute syn_noprune of STRUCT : architecture is true;
attribute syn_noprune of DCM_BASE : component is true;
attribute syn_black_box of DCM_BASE : component is true;


component bufg is
port (O :out std_logic; I : in std_logic);
end component;
attribute syn_noprune : boolean;
attribute syn_noprune of bufg: component is true;
begin

logic_0 <= '0';
zxl_clkh <= CLK_PSSCLK; –clk_int
zxl_clkm <=CLK_PSSCLK; –clk_int
zxl_clkidel <=clkfx_out_zxlsys3;
refo_clk <= CLK_PSSCLK;–clkfx_out_ref1;
vlo_clk <= CLKFX_OUT_PSSCLK1;–clkfx_out_vlo1;
modosc_clk <= clkfx_out_modclk1;
```

```
dco_clk <= CLKFX_OUT_DCO1;
pssclk_clk <= clkfx_out_pssclk1;
lfxt_clk <= clkfx_out_DCM11;
pgen_clk<=clkfx_out_pgen1;


CLK_int <= CLK_PAD;



USB : DCM_BASE generic map (
CLKDV_DIVIDE => 2.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 1, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 4, – Can be any Integer from 2 to 32
CLK_FEEDBACK => "2X", – Specify clock feedback of NONE or 1X
DLL_FREQUENCY_MODE => "HIGH",
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_SPEED") – LOW, HIGH, or HIGH_SER
frequency mode for DLL


port map (
CLKFB => CLK_USB,
–CLK90 => CLK90_tmp,
CLKIN => CLK_int,

RST => config_rst,
CLK0 => open,
CLKFX => CLKFX_OUT_tmp,

LOCKED => open,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>clk0_USB,
CLK2X180 => open ,
CLKDV => open,
CLKFX180 => open
);

PGEN : DCM_BASE generic map (
CLKDV_DIVIDE => 2.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
```

```
CLKFX_DIVIDE => 2, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 11,
CLK_FEEDBACK => "none",
DLL_FREQUENCY_MODE => "LOW",
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_Speed")

port map (
CLKFB =>open,
CLK0 => open,
CLKIN => CLK_int,
RST => config_rst,
CLKFX => CLKFX_OUT_pgentmp,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
CLKDV => open,
CLKFX180 => open
);

DCO : DCM_BASE generic map (
CLKDV_DIVIDE => 2.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 4, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 2,
CLK_FEEDBACK => "1X",
DLL_FREQUENCY_MODE => "LOW",
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_RANGE")
port map (
CLKFB =>CLK_DCO,
CLK0 => CLK0_DCO,
CLKIN => CLK_int,
RST => config_rst,
CLKFX => open,–CLKFX_OUT_dcotmp,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
```

```
CLKDV => CLKFX_OUT_dcotmp,–open,
CLKFX180 => open
);



MODCLK : DCM_BASE generic map (
CLKDV_DIVIDE => 5.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 24, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 5,
CLK_FEEDBACK => "1X",
DLL_FREQUENCY_MODE => "LOW",
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_RANGE")
port map (
CLKFB =>CLK_MODCLK,
CLK0 => CLK0_MODCLK,
CLKIN => CLK_int,
RST => config_rst,
CLKFX => open,
–DRDY => DRDY_out,
–DO => DO_out,
LOCKED => LOCKED_MODCLKtmp,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
CLKDV => CLKFX_OUT_modclktmp,
CLKFX180 => open
);



PSSCLK : DCM_BASE generic map (
CLKDV_DIVIDE => 16.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 4, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 2,
CLK_FEEDBACK => "1X",
DLL_FREQUENCY_MODE => "LOW",
```

```
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_RANGE")

port map (
CLKFB =>CLK_PSSCLK,
CLK0 => CLK0_PSSCLK,
CLKIN => clk_int,
RST => config_rst,–RST_INT,
CLKFX => open,–CLKFX_OUT_pssclktmp,
–DRDY => DRDY_out,
–DO => DO_out,
LOCKED => LOCKED_PSSCLKtmp,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
CLKDV =>CLKFX_OUT_pssclktmp,–open,
CLKFX180 => open
);

ZXLSYS_Inter : DCM_BASE generic map (
CLKDV_DIVIDE => 2.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 2, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 11,
CLK_FEEDBACK => "NONE",
DLL_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_SPEED")
port map (
CLKFB =>open,–CLK_ZXLSYS,
CLK0 => open,–CLK0_ZXLSYS,
CLKIN => CLK_int,
RST => config_rst,
CLKFX => CLKFX_OUT_zxlsystmp,

CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
CLKDV => open,
```

```
CLKFX180 => open,
–DRDY => DRDY_out,
–DO => DO_out,
LOCKED => zxl_lock
);

U13 : BUFG port map (I => CLKFX_OUT_zxlsystmp, O => CLKFX_OUT_zxlsys3);


LFXT : DCM_BASE generic map (
CLKDV_DIVIDE => 16.0, – Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
CLKFX_DIVIDE => 4, – Can be any interger from 1 to 32
CLKFX_MULTIPLY => 2,
CLK_FEEDBACK => "1X",
DLL_FREQUENCY_MODE => "LOW",
DFS_FREQUENCY_MODE => "LOW",
CLKIN_PERIOD => 41.667,
DCM_PERFORMANCE_MODE => "MAX_RANGE")

port map (
CLKFB =>CLK_DCM1,
CLK0 => CLK0_DCM1,
CLKIN => clk_int,
RST => config_rst,
CLKFX => open,–CLKFX_OUT_pssclktmp,
–DRDY => DRDY_out,
–DO => DO_out,
LOCKED => LOCKED_LFXT,
CLK90 => open,
CLK180 => open,
clk270 => open,
CLK2X =>open,
CLK2X180 => open ,
CLKDV =>CLKFX_OUT_DCM1tmp,
CLKFX180 => open
);




clkdiv_out_vlo <= clkdiv_out_ref;
```

```
process(clk_int)
variable count1 : integer := 0;
begin

if rising_edge(clk_int) then
if (count1<10) then
count1 := count1 +1;
else
config_rst<='0';
end if;
end if;
end process;

process(clk_int)
variable count2 : integer := 0;
begin

if rising_edge(clk_int) then
if (count2<72) then
count2 := count2 +1;
else
config_rst_bor<='0';
end if;
end if;
end process;



U3 : BUFG port map (I => clk0_USB, O => CLK_USB);
U4 : BUFG port map (I => CLKFX_OUT_tmp, O => CLK_tmp);
U6 : BUFG port map (I => CLKFX_OUT_pgentmp, O => CLKFX_OUT_pgen1);
U7 : BUFG port map (I => CLKFX_OUT_dcotmp, O => CLKFX_OUT_DCO1);
U10 : BUFG port map (I => CLKFX_OUT_MODCLKtmp, O => CLKFX_OUT_modclk1);
U12 : BUFG port map (I => CLKFX_OUT_pssclktmp, O => CLKFX_OUT_pssclk1);
U15 : BUFG port map (I => CLK0_DCO, O => CLK_DCO);
U18 : BUFG port map (I => CLK0_MODCLK, O => CLK_MODCLK);
U20 : BUFG port map (I => clk0_PSSCLK, O => CLK_PSSCLK);
U22 : BUFG port map (I => clkdiv_out_vlo, O => clkfx_out_vlo1);
U24 : BUFG port map (I => clkdiv_out_ref, O => clkfx_out_ref1);
U25 : BUFG port map (I => clk0_DCM1, O => CLK_DCM1);
U26 : BUFG port map (I => clkFX_OUT_DCM1TMP, O => CLKFX_OUT_DCM11);
U29 : BUF port map (I => locked_modclktmp, O => locked_modclk);
```

```
U30 : BUF port map (I => locked_pssclktmp, O => locked_pssclk);



CLKFX_OUT_PSSCLK <= CLKFX_OUT_PSSCLK1;
CLKFX_OUT_PGEN <= CLKFX_OUT_PGEN1;
CLKFX_OUT_MODCLK <= CLKFX_OUT_MODCLK1;
CLKFX_OUT_ZXLSYS <= clkfx_out_zxlsys3;
CLKFX_OUT_DCO <= CLKFX_OUT_DCO1;
CLKFX_OUT_VLO <= CLKFX_OUT_PSSCLK1;–CLKFX_OUT_VLO1;
CLKFX_OUT_REF <= CLK_PSSCLK;–CLKFX_OUT_REF1;
CLKFX_OUT_lfxt <= CLKFX_OUT_DCM11;
CLKOUT_zxlmh <= CLK_PSSCLK;
config_rst1 <= config_rst_bor;
clk_out <= clk_int;

end architecture STRUCT;
```

A.4 BRAM instantiation

```
----------------------------------------------------------------------
– Title : BRAM emulation of flash for MSP430F5172 prototype
– Description: Verilog file generated for memory of 32x1024 bits.memory depth can be
changed by modifying the addrwidth parameter. data width can be changed by modi-
fying the datawidth parameter.Virtex5 LX110 has a maximum of 128 BRAMs. verily
wrapper should be written to instantiate this memory.

–Date :2010 july 9
----------------------------------------------------------------------


‘timescale 1ns/100ps
module flash1k

(
PortACLK
,PortAAddr


,PortBClk
,PortBDataIn
```

```
,PortBAddr

,PortADataOut

)/* synthesis syn_noprune=1 */;

parameter DATAWIDTH = 32;
parameter ADDRWIDTH = 10;
parameter MEMDEPTH = 2**(ADDRWIDTH);

parameter SPRAM = 0;
parameter READ_MODE_A = 1;
parameter READ_WRITE_A = 2;
parameter ENABLE_WR_PORTA = 0;

parameter REGISTER_RD_ADDR_PORTA = 0;

parameter REGISTER_OUTPUT_PORTA = 1;
parameter ENABLE_OUTPUT_REG_PORTA = 0;
parameter RESET_OUTPUT_REG_PORTA = 0;

parameter READ_MODE_B = 1;
parameter READ_WRITE_B = 3;
parameter ENABLE_WR_PORTB = 0;

parameter REGISTER_RD_ADDR_PORTB = 0;

parameter REGISTER_OUTPUT_PORTB = 0;
parameter ENABLE_OUTPUT_REG_PORTB = 0;
parameter RESET_OUTPUT_REG_PORTB = 0;

input PortACLK;
```

```
input [ADDRWIDTH-1:0] PortAAddr;


input PortBClk;
input [DATAWIDTH-1:0] PortBDataIn;
input [ADDRWIDTH-1:0] PortBAddr;


output [DATAWIDTH-1:0] PortADataOut;



wire PortACLK;
wire [ADDRWIDTH-1:0] PortAAddr;
wire [DATAWIDTH-1:0] PortADataIn;


wire [DATAWIDTH-1:0] PortADataOut;


wire PortAWriteEnable;
wire PortAReadEnable;
wire PortAReset;


wire PortBClk;
wire [DATAWIDTH-1:0] PortBDataIn;
wire PortBWriteEnable;
wire [ADDRWIDTH-1:0] PortBAddr;


wire [DATAWIDTH-1:0] PortBDataOut;


wire PortBReadEnable;
wire PortBReset;



Syncore_ram
#(
.SPRAM(SPRAM)
,.READ_MODE_A(READ_MODE_A)
```

,.READ_MODE_B(READ_MODE_B)
,.READ_WRITE_A(READ_WRITE_A)
,.READ_WRITE_B(READ_WRITE_B)
,.DATAWIDTH(DATAWIDTH)
,.ADDRWIDTH(ADDRWIDTH)
,.ENABLE_WR_PORTA(ENABLE_WR_PORTA)
,.REGISTER_RD_ADDR_PORTA(REGISTER_RD_ADDR_PORTA)
,.REGISTER_OUTPUT_PORTA(REGISTER_OUTPUT_PORTA)
,.ENABLE_OUTPUT_REG_PORTA(ENABLE_OUTPUT_REG_PORTA)
,.RESET_OUTPUT_REG_PORTA(RESET_OUTPUT_REG_PORTA)
,.ENABLE_WR_PORTB(ENABLE_WR_PORTB)
,.REGISTER_RD_ADDR_PORTB(REGISTER_RD_ADDR_PORTB)
,.REGISTER_OUTPUT_PORTB(REGISTER_OUTPUT_PORTB)
,.ENABLE_OUTPUT_REG_PORTB(ENABLE_OUTPUT_REG_PORTB)
,.RESET_OUTPUT_REG_PORTB(RESET_OUTPUT_REG_PORTB)
)
U3(
.PortClk(PortBClk, PortACLK)
,.PortReset(PortBReset, PortAReset)
,.PortWriteEnable(PortBWriteEnable, PortAWriteEnable)
,.PortReadEnable(PortBReadEnable, PortAReadEnable)
,.PortDataIn(PortBDataIn, PortADataIn)
,.PortAddr(PortBAddr, PortAAddr)
,.PortDataOut(PortBDataOut, PortADataOut)
);
endmodule

Appendix B FPGA editor sample scripts

B.1 Probes scripts

– Title : script file for probes
– Description: The script file can be used as a template to route internal signals to unused io pins on the FPGA for probing internal signals of the design.
–Date :2010 july 9

---

post probes
probe add u_dig_top/u_flash_mem/N_176_0_i -targetpins AV41 -noroute
probe route u_dig_top/u_flash_mem/N_176_0_i

probe add u_dig_top/u_flash_mem/N_177_0_i -targetpins AU42 -noroute
probe route u_dig_top/u_flash_mem/N_177_0_i
probe add u_dig_top/u_flash_mem/N_178_0_i -targetpins AU41 -noroute
probe route u_dig_top/u_flash_mem/N_178_0_i
probe add u_dig_top/u_flash_mem/N_179_0_i -targetpins AT41 -noroute
probe route u_dig_top/u_flash_mem/N_179_0_i
probe add u_dig_top/u_flash_mem/N_180_0_i -targetpins AT42 -noroute
probe route u_dig_top/u_flash_mem/N_180_0_i
probe add u_dig_top/u_flash_mem/N_181_0_i -targetpins AR42 -noroute
probe route u_dig_top/u_flash_mem/N_181_0_i
probe add u_dig_top/u_flash_mem/N_182_0_i -targetpins AP41 -noroute
probe route u_dig_top/u_flash_mem/N_182_0_i
probe add u_dig_top/u_flash_mem/N_183_0_i -targetpins AP42 -noroute
probe route u_dig_top/u_flash_mem/N_183_0_i
probe add u_dig_top/u_flash_mem/N_184_0_i -targetpins AN41 -noroute
probe route u_dig_top/u_flash_mem/N_184_0_i
probe add u_dig_top/u_flash_mem/N_185_0_i -targetpins AM41 -noroute
probe route u_dig_top/u_flash_mem/N_185_0_i
probe add u_dig_top/u_flash_mem/N_186_0_i -targetpins AM42 -noroute
probe route u_dig_top/u_flash_mem/N_186_0_i
probe add u_dig_top/u_flash_mem/N_187_0_i -targetpins AL42 -noroute
probe route u_dig_top/u_flash_mem/N_187_0_i
probe add u_dig_top/u_flash_mem/N_188_0_i -targetpins AK42 -noroute
probe route u_dig_top/u_flash_mem/N_188_0_i
probe add u_dig_top/u_flash_mem/N_189_0_i -targetpins AL41 -noroute
probe route u_dig_top/u_flash_mem/N_189_0_i
probe add u_dig_top/u_flash_mem/N_190_0_i -targetpins AL40 -noroute
probe route u_dig_top/u_flash_mem/N_190_0_i
probe add u_dig_top/u_flash_mem/N_191_0_i -targetpins AK40 -noroute
probe route u_dig_top/u_flash_mem/N_191_0_i
probe add u_dig_top/u_flash_mem/N_192_0_i -targetpins AC39 -noroute
probe route u_dig_top/u_flash_mem/N_192_0_i
probe add u_dig_top/u_flash_mem/N_193_0_i -targetpins AC40 -noroute
probe route u_dig_top/u_flash_mem/N_193_0_i
probe add u_dig_top/u_flash_mem/N_194_0_i -targetpins AD40 -noroute
probe route u_dig_top/u_flash_mem/N_194_0_i
probe add u_dig_top/u_flash_mem/N_195_0_i -targetpins AE40 -noroute
probe route u_dig_top/u_flash_mem/N_195_0_i
probe add u_dig_top/u_flash_mem/N_196_0_i -targetpins AH41 -noroute
probe route u_dig_top/u_flash_mem/N_196_0_i
probe add u_dig_top/u_flash_mem/N_197_0_i -targetpins Ag42 -noroute
probe route u_dig_top/u_flash_mem/N_197_0_i
probe add u_dig_top/u_flash_mem/N_198_0_i -targetpins Ag41 -noroute

```
probe route u_dig_top/u_flash_mem/N_198_0_i
probe add u_dig_top/u_flash_mem/N_199_0_i -targetpins Af40 -noroute
probe route u_dig_top/u_flash_mem/N_199_0_i
probe add u_dig_top/u_flash_mem/N_200_0_i -targetpins Af42 -noroute
probe route u_dig_top/u_flash_mem/N_200_0_i
probe add u_dig_top/u_flash_mem/N_201_0_i -targetpins Af41 -noroute
probe route u_dig_top/u_flash_mem/N_201_0_i
probe add u_dig_top/u_flash_mem/N_202_0_i -targetpins Ad41 -noroute
probe route u_dig_top/u_flash_mem/N_202_0_i
probe add u_dig_top/u_flash_mem/N_203_0_i -targetpins Ae42 -noroute
probe route u_dig_top/u_flash_mem/N_203_0_i
probe add u_dig_top/u_flash_mem/N_204_0_i -targetpins Ad42 -noroute
probe route u_dig_top/u_flash_mem/N_204_0_i
probe add u_dig_top/u_flash_mem/N_205_0_i -targetpins Ac41 -noroute
probe route u_dig_top/u_flash_mem/N_205_0_i
probe add u_dig_top/u_flash_mem/N_206_0_i -targetpins Ab42 -noroute
probe route u_dig_top/u_flash_mem/N_206_0_i
probe add u_dig_top/u_flash_mem/N_207_0_i -targetpins Ab41 -noroute
probe route u_dig_top/u_flash_mem/N_207_0_i
end
```

Appendix C BRAM initialisation script

– Title : BRAM memory initialization script. for configuring device id into the info flash memory.
– Description: The script file is used to initialize the device id into the info flash memory and also emulates the ”all ones” initial condition of flash. the same template can be used to configure other memories in the FPGA.
–Date :2010 july 9

```
unselect -all
select comp 'u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0'
unselect -all
select comp 'u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0'
post block
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_00
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_01
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
```

setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_02
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_03
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_04
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_05
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_06
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_07
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_08
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_09
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_0F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_10
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00008030FFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_11
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_12
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_13
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_14
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_15
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_16
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_17

“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_18
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_19
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_1F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_20
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_21
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_22
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_23
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_24
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_25
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_26
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_27
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_28
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_29
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”

setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_2F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_30
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_31
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_32
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_33
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_34
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_35
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_36
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_37
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_38
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_39
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_3F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_40
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_41
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_42

“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_43
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_44
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_45
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_46
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_47
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_48
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_49
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_4F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_50
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_51
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_52
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_53
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_54
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_55
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_56
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_57
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”

setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_58
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_59
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_5F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_60
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_61
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_62
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_63
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_64
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_65
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_66
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_67
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_68
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_69
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6D

“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_6F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_70
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_71
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_72
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_73
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_74
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_75
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_76
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_77
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_78
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_79
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7A
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7B
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7C
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7D
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7E
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”
setattr comp u_dig_top/u_flash_mem/infoflash/u9/U3/mem_mem_0_0 INIT_7F
“FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”

end

Appendix D FPGA prototype test sample code

```
————————————————————————————————————
– Title : low power mode test code template for MSP430F5172 prototype
– Description: The code configures the device in port1 interrupt mode specifically on pin p1.4 and p1.1 is toggled
every time an interrupt occurs on p1.4. the port 1 interrupt is used to remove the device out of low power mode.
–Date :2010 july 9
————————————————————————————————————

#include "MSP430F5172.h"

void main(void)

volatile unsigned int i;
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
UCSCTL4 = SELA1 +SELA0;
P3DIR = BIT4+BIT2; //set direction
P3SEL = BIT2+BIT4; // Set P3.2,3.4 as clk outputs
P1DIR —= 0x01; // Set P1.0 to output direction
P1REN —= 0x10; // Enable P1.4 internal resistance
P1OUT —= 0x10; // Set P1.4 as pull-Up resistance
P1IE —= 0x10; // P1.4 interrupt enabled
P1IES —= 0x10; // P1.4 Hi/Lo edge
P1IFG &= 0x10; // P1.4 IFG cleared

while(1) //infinite loop


__bis_SR_register(LPM1_bits + GIE); // Enter LPM4 w/interrupt
for (i = 200; i >0; i–); // Delay




// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)

__bic_SR_register_on_exit(LPM1_bits + GIE);
```

P1OUT ^= 0x01; // P1.0 = toggle
P1IFG &= 0x010; // P1.4 IFG cleared

References

[1] Altera. Asic prototyping with stratix seies fpga.

[2] Altera. Timing analysis of internally generated clocks in timequest. Technical report, 2009.

[3] Xilinx AR18181. http://www.xilinx.com/support/answers/18181.htm, august 2007.

[4] Synopsys. Virtex 5 synthesis issue. synopsys case id 8000395645.

[5] Synopsys. Designware Building Block IP User Guide, June 2009.

[6] Synplicity. *Synplify Pro-User Guide and Tutorial*, February 2001.

[7] Elgris Technologies. www.elgris.com/content/edif_overview.html.

[8] Texas Instruments. *msp430f5xx family user guide*, july 2010.

[9] Xilinx. *Development system reference guide*, 12.1 edition.

[10] Xilinx. http://www.xilinx.com/itp/xilinx10/isehelp/pp-db-place-and-route-properties.htm.

[11] Xilinx. Advanced Xilinx FPGA Design with ISE, 2002.

[12] Xilinx. *Xilinx constraint user guide*, December 2008.

[13] Xilinx. *FPGA Editor user guide*, august 2009.

[14] Xilinx. *Virtex 5 FPGA configuration guide*, august 2009.

[15] Xilinx. *Virtex 5 user guide*, November 2009.

[16] Xilinx. *Design and analysis with Planahead - Fpga days*, 2010.

[17] Xilinx. http://63.241.181.135/itp/xilinx10/isehelp/ise_c_design_strategies.htm, January 2010.

[18] Xilinx. www.xilinx.com/itp/xilinx10/isehelp/pce_p_registers_in_job.htm, January 2010.