CME SPAN Algorithm on FPGA.

Pramod Nayak (ppn2106@columbia.edu), Ankit Pradhan (ap3188@columbia.edu),
Vidhatre Gathey (vvg2111@columbia.edu),
Bhargav Sethuram (bs2814@columbia.edu).

Guided by

Prof.David Lariviere
Prof. Stephen Edwards
Department of Computer Science,
Columbia University,
New York

May 14, 2014

1 Abstract

The Standard Portfolio Analysis of Risk (SPAN) margin system, proposed by the Chicago Mercantile Exchange provides a method to integrate both futures and options on futures contracts into the same system to assess a portfolio's risk. In the SPAN Methodology, the contracts are examined over a range of price and volatility changes to determine potential gains and losses. SPAN also allows for both Inter-Month and Inter-Commodity spreading among the different commodities in the portfolio. There is a need for real time risk monitoring systems, both by exchanges & by the client members to keep a check on the high speeds at which trading is executed currently. While the risk monitoring by the exchange has a linear input, the currently matched orders in the portfolio, the pre-trade risk monitoring calculation is combinatorial in nature. With an order flow of 1000 orders per second, it is imperative to shift to hardware based infrastructure to support the current low latency trading infrastructure & its pre –trade risk check.

Our project aims to explore the SPAN Algorithm for computing the initial margin, keeping in mind the latency requirements of the exchanges and the exchange members. The main focus of the project is to implement the industry standard, SPAN Algorithm on a Hardware Platform to suit the demands of the pre-trade risk monitoring in the High-Frequency Trading Environment. The Hardware Based Implementation will be performed on randomized portfolios of futures & options on futures contracts, and the outputs will be analyzed for accuracy and timing analysis.

2 Overview

Motivation

Just as in the day-to-day uncertainties we face pertaining to weather, health, traffic etc, the stock markets face an uncertainty in the movement of share prices.

Futures are standardized contracts for the purchase and sale of financial instruments or physical commodities for future delivery on a regulated commodity futures exchange.

A futures contract allows a trader to undertake a contract to accept or make delivery of a commodity or some kind of financial asset essentially in the

(a) in the future on a known date,

(b) under specified conditions,

(c) for a price contracted today.

In the futures market, the party to contract, which agrees to take delivery of the commodity, is long in the position, whereas the firm, which has to deliver the commodity, is short in position. A speculator will benefit in the futures market is he is long and the prices rise, and shorts if the price falls.

The exchange or the clearing firm, through the submission of bids and asks will match long orders with short orders, either with outside traders or with their own trades.

Due to the changing demands for futures services and the costs of doing business, which in turn were related to changes in the general economy and in agriculture especially, gave rise to the concept of the Initial Margin Requirements. The current futures and options contracts require a complex margining system. The value of a contract is “marked to the market” each day (or a few times a day), which means that losses and gains related to the changing value of the contract are settled by the end of the day. In the Trading Environments, traders are financially responsible to the clearing house or the exchange, if they are a member of the clearing organization or the exchange or indirectly if the trade goes through a Commission Merchant. Margins are an important component of the institutional arrangements that help ensure the integrity of futures and options contracts. The margin rates are set by the exchanges and some brokerage will add an extra premium to the exchange minimum rate in order to lower their risk exposure. The Initial Margin is set based on the risk associated with the commodity.

The positions in the futures and options market have legal obligation to make or take delivery and margins can be considered as security deposits (performance bonds), to ensure the performance on the contract. The buyer receives the underlying asset and makes the complete payments when the seller makes the delivery or its equivalent cash settlements. Since in the current trends, futures are cancelled by taking opposite positions rather than delivery of the actual commodity and since both short and long positions need to be margined, the initial margins should not be considered as a down payment for the contract.

In finance, a margin is the collateral that the holder of a financial instrument has to deposit to cover some or all of the credit risk of their counterparty, mostly a broker or an exchange. The collateral can be in the

form of cash or securities, and it is deposited in a margin account. Options and futures traders are required to have a certain amount of margins in their accounts to cover for the potential losses. The SPAN (Standard Portfolio Analysis of Risk) methodology developed by CME is used worldwide to calculate margins on options & futures. The SPAN Methodology uses complex algorithms and sets margin of each position to its calculated worst possible one-day move. The system after calculating the margin of each position can shift any excess margin on existing positions to new positions or existing positions that are short of margin.

SPAN calculates margin for a portfolio of positions based on margin parameters determined by individual exchanges/clearinghouses. Therefore, identical futures contracts traded on more than one exchange may have different SPAN-calculated margin requirements.

Minimum margins are calculated in SPAN by the determination of appropriate parameters, such as margin scan ranges and volatility scan ranges, for each underlying futures contract traded on the exchange. An exchange may elect to change its margin requirements as often as daily, or may never change them after they have been initially set if the underlying contract price is stable.

The overall portfolio risk is calculated by evaluating the worst possible loss that instruments in a portfolio may incur over a trading day. This is done by computing the gains and losses of portfolio, influenced by the various market conditions. The SPAN risk array, which is a set of numerical values, indicates a particular contract gaining or losing value under various conditions. Each condition is called a risk scenario. The numeric value for each risk scenario represents the gain or loss that that particular contract will experience for a particular combination of price (or underlying price) change, volatility change, and decrease in time to expiration.

The SPAN margin files sent out by the exchange to the organizations implementing SPAN, and are plugged into a SPAN margin calculator. For the futures options, they are assumed to have risk until they expire out of account or are closed. SPAN takes into account all the market scenarios and cases of extreme market volatility, to evaluate the margin impact of these futures options. The SPAN margin requirements are compared against broker's pre-defined extreme market move scenarios and the greater of the two are utilized as margin requirement.

3 SPAN Algorithm

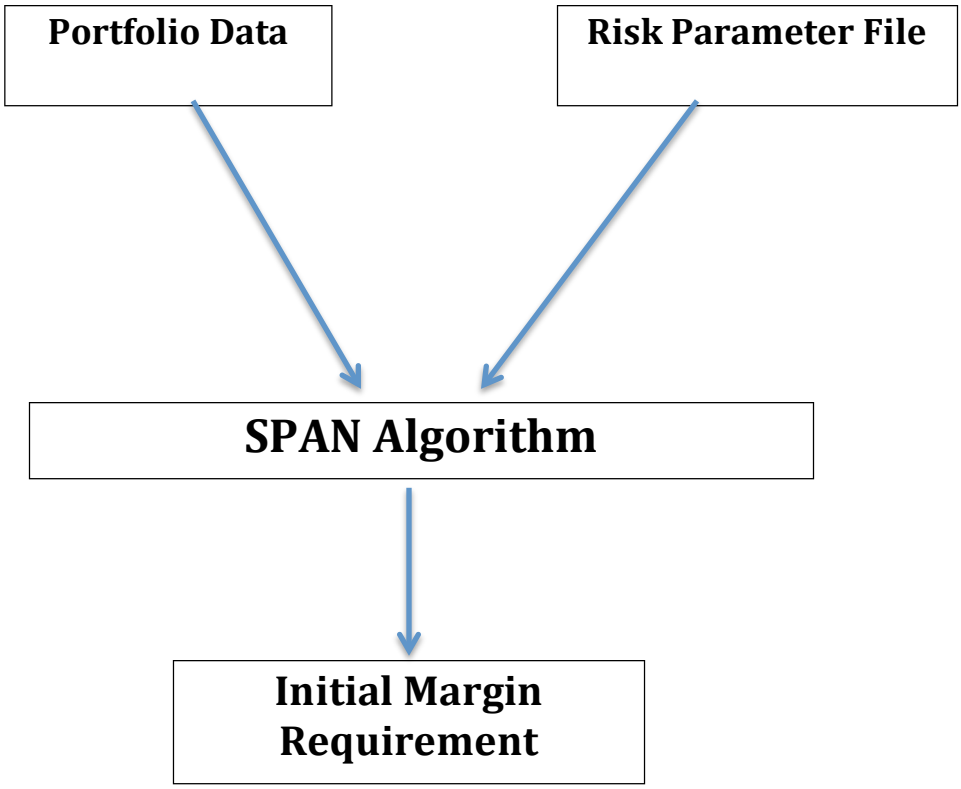
The algorithm is made up of four modules:

1) Scanning Risk
2) Intra Commodity Spread Credit
3) Inter Commodity Spread Credit
4) Short Option Charge.

The SPAN Algorithm for a portfolio containing a combined commodity is calculated as follows –

1) Sum of risk, the Intracommodity Spread Risk and the Delivery Spot Risk
2) Subtracting the Inter Commodity Spread Credit from the above.
3) Taking the maximum value of the result and the short option minimum.

The individual modules making up the SPAN Algorithm are discussed below with examples. Much time was spent in understanding the complex algorithm & identifying corner cases to make the implementation robust. To calculate the Initial Margin for the portfolio, the modules are calculated in parallel, as they are computed independent of each other. The top-level block diagram of the module is shown in Fig 1.



Block Diagram of the SPAN Algorithm

3.1 Scanning Risk

The first calculation in SPAN is the Scanning Risk, and it is performed on a combined commodity level assuming perfect correlations in price and volatility movements of the underlying instruments over time. Each bin of orders in the portfolio with the same underlying asset is subjected to a series of 16 different risk scenarios, where two parameters are used: the price scan range and volatility scan range.

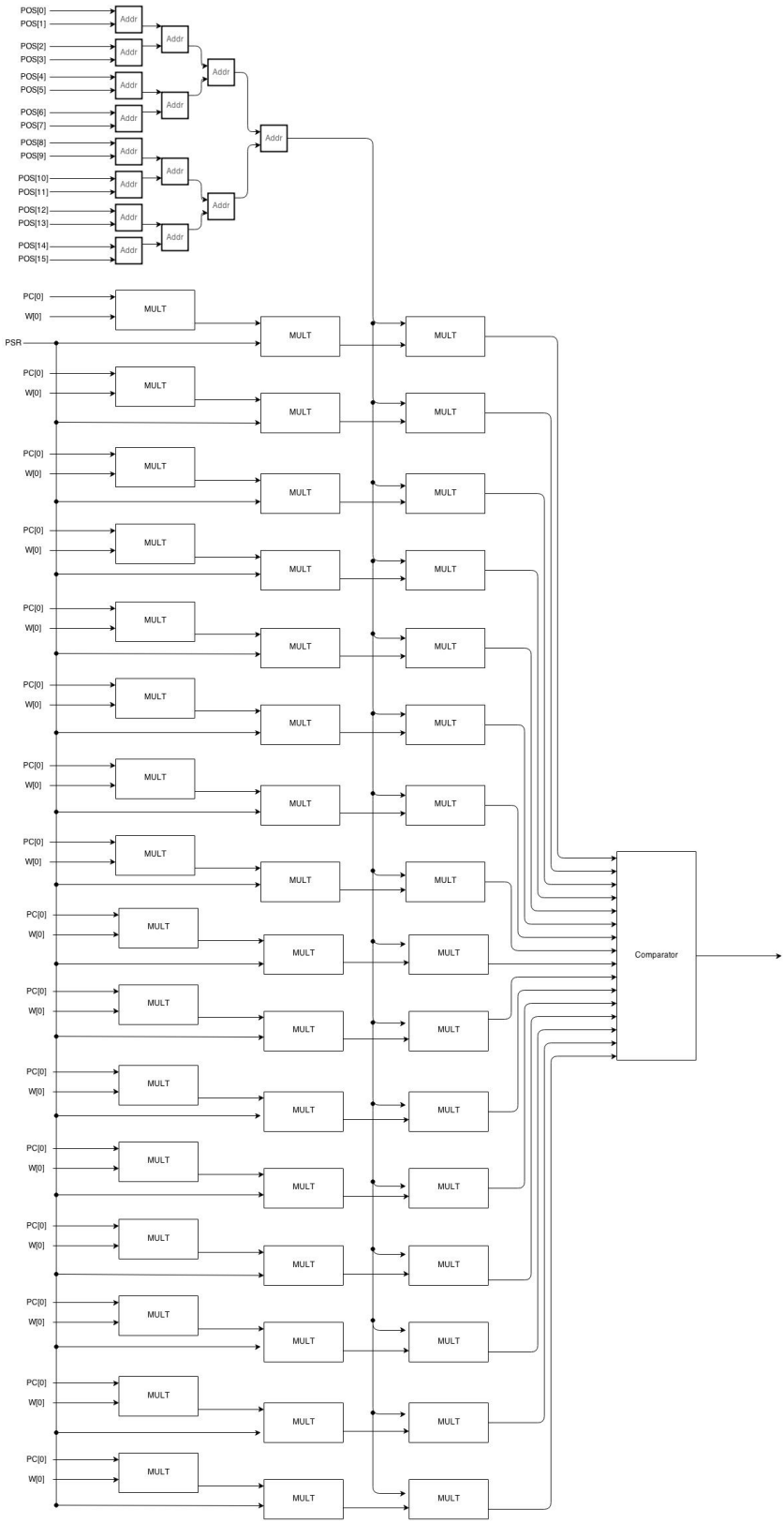
Each combined commodity can consist of several futures contracts and options, each with a different position. To calculate the Scanning Risk for the combined commodity, each order has its associated risk array multiplied by its position, and then the value changes of all order in each risk scenario are summed together. The risk scenario with the highest value, indicating the conditions under which the combined commodity will experience the highest possible loss, is then chosen as the Active Scenario, and the associated loss is set as the Scanning Risk.

The Scanning Risk, in other words, is just the worst-case outcome of the stress tests in the risk array.

The 16 risk scenarios are all different combinations of movements in price & implied volatility futures contracts, with applied weights to vary probabilities for these movements. The two extreme scenarios, scenarios 15 and 16, consist of drastic price movements, but their low probabilities of occurring are reflected in the lower weights placed on them. When applied to a futures contract or option, each risk scenario will yield the value loss for that order at the given price and volatility movements. For instance, a long futures contract under risk scenario 10 will experience a value loss of two thirds its price scan range, whereas a short futures contract in the same scenario would experience a value gain of the same amount, indicated by a negative value loss.

Scenario	SP Underlying Price Move	Volatility Move	SP Future Gain/Loss	SP Option Gain/Loss	Portfolio Gain/Loss
1	UNCHANGED	UP	0	1,807	1807
2	UNCHANGED	DOWN	0	-1,838	-1,838
3	UP 33%	UP	-7,499	7,899	400
4	UP 33%	DOWN	-7,499	5,061	-2,438
5	DOWN 33%	UP	7,499	-3,836	3,663
6	DOWN 33%	DOWN	7,499	-8,260	-761
7	UP 67%	UP	-15,001	14,360	-641
8	UP 67%	DOWN	-15,001	12,253	-2,748
9	DOWN 67%	UP	15,001	-8,949	6,052
10	DOWN 67%	DOWN	15,001	-13,980	1,021
11	UP 100%	UP	-22,500	21,107	-1,393
12	UP 100%	DOWN	-22,500	19,604	-2,896
13	DOWN 100%	UP	22,500	-13,455	9,045
14	DOWN 100%	DOWN	22,500	-18,768	3,732
15	UP 300%	UNCHANGED	-22,275	21,288	-987
16	DOWN 300%	UNCHANGED	22,275	-9,160	13,115
				Largest Potential Loss = SPAN Risk	**13,115**

Reference: http://www.cmegroup.com/clearing/files/span-methodology.pdf



The block diagram above shows how the Price Scan Range is computes for a portfolio.

The Operation of the Scanning Risk Module can be described in short as follows:

- Read input parameters and portfolio data
- Calculate price change for each of the 16 scenarios.
- (Price Scan Range(PSR) * Price Change(PC))
- Multiplies by weight (reference to probability of event)
- Choose maximum price change of asset among these scenarios.

3.2 Intracommodity Spread Charge

The second parameter computed is the Intra-Commodity Spread Credit, which evaluates the basis risk between contracts with different expirations within the same commodity, where there is imperfect correlation of price and volatility movements over time, and allows precise targeting of these requirements to particular intracommodity strategies.

While the portfolio under consideration consists of two orders with different maturities eg: 3 months for the futures A contract and 2 months for the futures contract B- the price movements of these orders are considered to be perfectly correlated in the Scanning Risk step. In each risk scenario, all prices move in the same direction and by the same amount simultaneously. In other words, the Scanning Risk calculation does not account for the fact that prices of orders with different maturities respond differently to changing market conditions. Since futures prices do not correlate exactly across contract months, a gain in one month may not exactly offset losses in another month.

For a particular portfolio, various Tiers & Spread Charge associated with the Tiers are assumed to calculate the Intra- Commodity Spread Credit.

Tiers
The combined commodity is first divided into tiers, where each tier contains orders with a preset range of maturities.

Tier	Maturity
1	0-1 months
2	1-2 months
3	2-3 months

The Tier Spread Table then sets the fixed costs of having spreads between different tiers in the combined commodity. These charges are typically set by the exchange and are dependent on the underlying asset of the combined commodity. To decide which spreads get what charge applied to them and in what order, a Spread Priority Table is also formed.

Tier Spread Table

Tier	Maturity	Tier 1	Tier 2	Tier 3
Tier 1	0-1 months	50 USD	X	X
Tier 2	1-2 months	80 USD	60 USD	X
Tier 3	2-3 months	90 USD	100 USD	70 USD

Tier Spread Table:

Priority	1	2	3	4	5	6
Tier Spread	1 to 1	2 to 2	3 to 3	1 to 2	1 to 3	2 to 3

Outright Charge:

Outright Charge for Tier 1: 180 USD

Outright Charge for Tier 2: 150 USD

Outright Charge for Tier 3: 100 USD

Example:

Example Portfolio:

Instrument	Futures	Futures	Futures
Position	10	15	-5
Maturity	90	25	150
Position Delta	10	15	-5

The delta spread table for the above portfolio is as follows:

Tier	Long	Short
1	15	0
2	10	0
3	0	-5

Consulting the Delta Spread Table, between Tier 1 & Tier 3, 5 Inter-tier spreads can be formed.

The process is continues until all possible spread formation between different Tiers are checked.

Hence, the Spread Margin = (Outright of Tier 1 – Outright of Tier 2) + (Number of Intermonth Spreads* Tier Spread Charge). = (180-150) + (5* 90) = 480 USD.

For larger combined commodities that contain orders with longer maturities, more tiers are formed to accommodate these orders, but the general method of calculation remains the same.

Hence, the Initial Margin Requirement for a portfolio containing futures: Scanning Risk + Intra Commodity Spread Credit Inter Commodity Spread Credit (Cross Commodity Spread Credit).

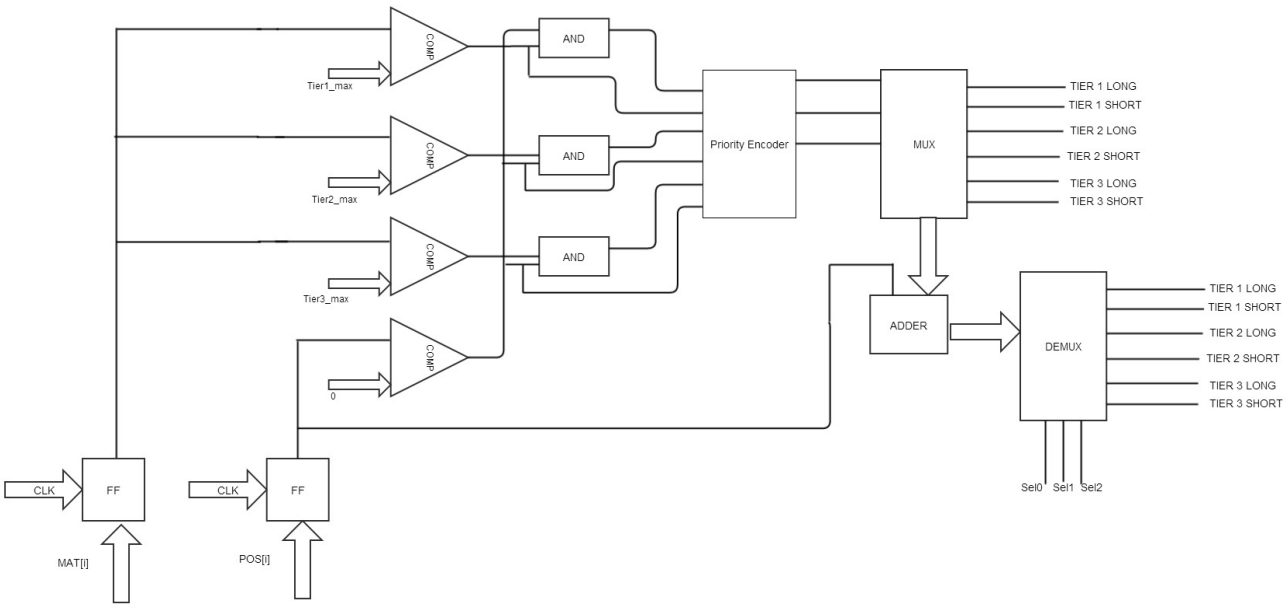
The operation of the InterMonth Spread Charge can be summarized as follows –

- Read the input portfolio data and SPAN tier information
- Create a standard Tier table based on maturity dates
- Sort the contracts according to the standard tier table
- Calculate the spreads for each tier pair combination (relative difference between long and short contracts)
- Multiply with the spread charge
- Add outright charge associated.

The block diagram for the InterMonth Spread Charge can be shown as –

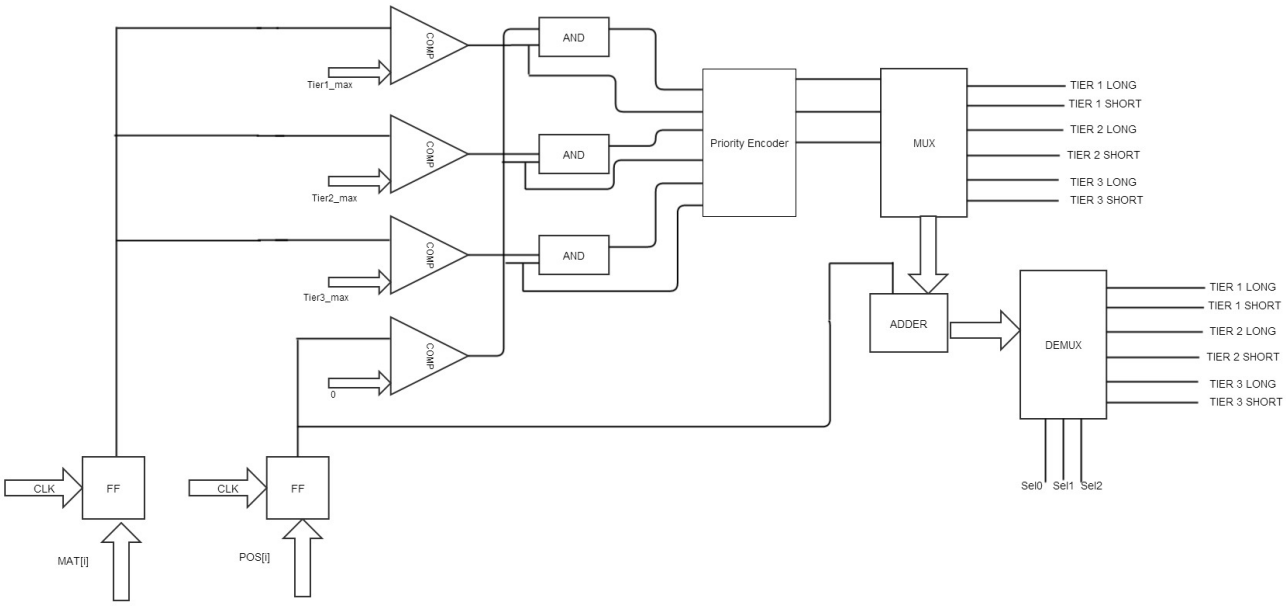
The InterMonth Spread Charge Computation Block can be divided into two parts –

1) To form the Long Short Table according to their maturity.(Figure shown below)
2) To compute the Tier Spread Charge between different Tiers. (Figure shown below).



The above block shows the computation of the Long Short Table into Tiers depending on the maturity of the contracts.

The block diagram shows how the Tier Spread Charge for each of the Tiers is computed.

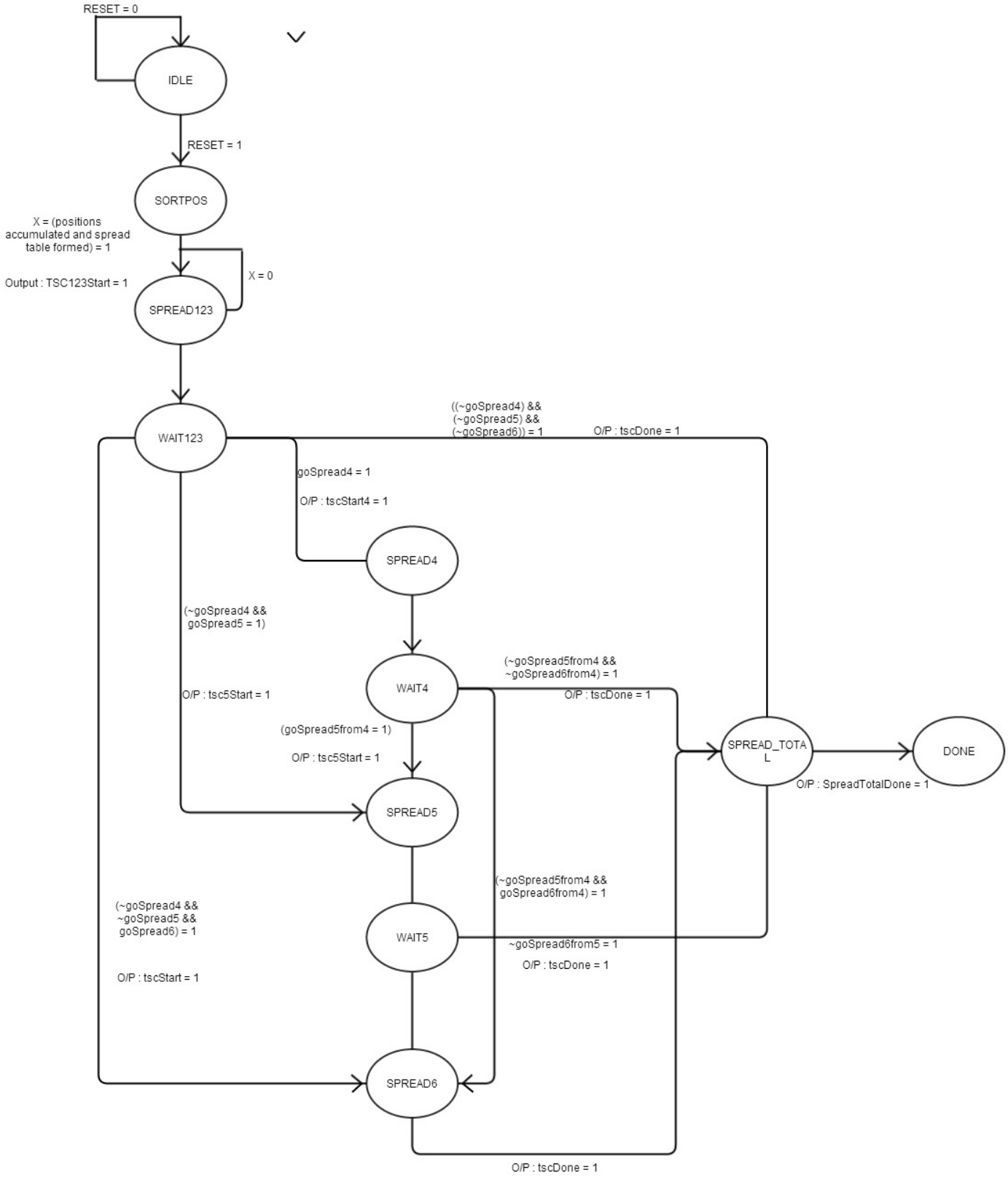


For the computation of the Tier Spread Charges between different Tiers, a particular priority order is followed as given in the Tier Priority Table mentioned above.

The Tier Spread Charge between Tier 1, Tier 2 and Tier 3 among themselves can be computed in parallel. While the Tier Spread Charges between, Tier 1 & Tier 2, Tier 1 & Tier 3 and Tier 2 and Tier 3, have to be computed sequentially and therefore a FSM was also built to be perform the tasks sequentially.

The FSM has been described in detail in the Source Code Section in the Appendix.

The FSM can be shown as follows :


RESET = 0
IDLE
RESET = 1
SORTPOS
X = (positions accumulated and spread table formed) = 1
Output : TSC123Start = 1
X = 0
SPREAD123
WAIT123
((~goSpread4) && (~goSpread5) && (~goSpread6)) = 1
O/P : tscDone = 1
goSpread4 = 1
O/P : tscStart4 = 1
SPREAD4
(~goSpread4 && goSpread5 = 1)
O/P : tsc5Start = 1
WAIT4
(~goSpread5from4 && ~goSpread6from4) = 1
O/P : tscDone = 1
(goSpread5from4 = 1)
O/P : tsc5Start = 1
SPREAD5
SPREAD_TOTAL
DONE
O/P : SpreadTotalDone = 1
(~goSpread5from4 && goSpread6from4) = 1
(~goSpread4 && ~goSpread5 && goSpread6) = 1
WAIT5
~goSpread6from5 = 1
O/P : tscDone = 1
O/P : tscStart = 1
SPREAD6
O/P : tscDone = 1

3.3 Cross Commodity Spread Credit

In order to recognize the risk reducing aspects of the portfolio's containing multiple commodities containing offsetting positions in highly correlated instruments, the SPAN algorithm forms the Inter-Commodity Spread Credit.

To recognize the risk reducing aspects of portfolios containing offsetting positions in highly correlated instruments, SPAN forms Inter-Commodity Spreads. The Inter-Commodity Spreads formed reduces the overall performance bond or margin requirement of the portfolio.

The Inter-Commodity Spread Credit needs to be taken into account for two different assets, which have correlation between them can have an offsetting effect on the overall risk exposure to the portfolio.

For example, if the exchange considers the price of gold to be positively related to the price of silver, a spread credit rate on the opposing positions in gold & silver is set. This takes into account that the losses in the gold long position due to decrease in the gold price, is partially offset by the gains in the short position on silver, due to accompanying decrease in the silver prices. Thus a portfolio with a long position in gold & a short position in silver would thus have its overall margin requirement reduced.

The Inter-Commodity Spread Credits are formed taking into account:

1) Which products are related, thereby, authorizing margin reduction for spread positions;

2) The ratio of positions that must be present in an account for the spread to be applied;

3) The amount of the spread credit; and

4) The priority for applying spreads.

Also it must be noted that the Inter-Commodity Spread Credit will be zero for a portfolio containing only one combined commodity as no spreads can be formed, as there is no correlated commodity in the portfolio.

For example, considering a portfolio consisting of Gold & Silver, with an Inter Rate of 60 %.

Gold vs Silver (2:1) – 55 % Inter Rate.

Outright Rates

Gold $175, Silver $250

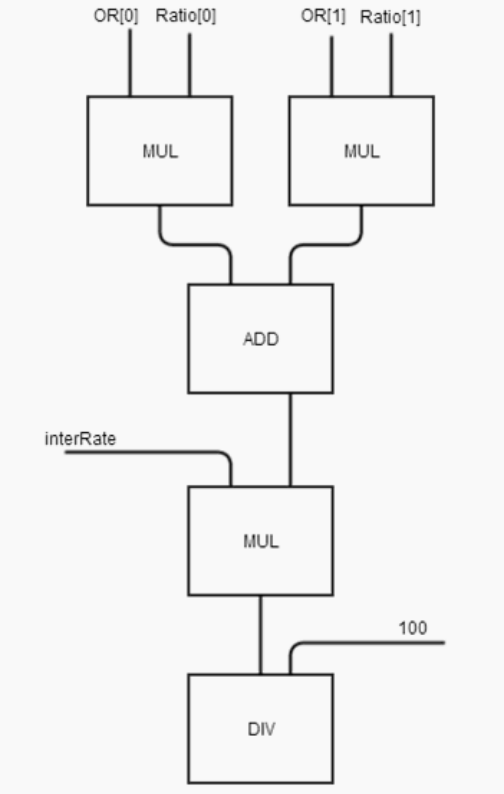
The outright margin before the Inter Spread Credit is – ($175*2) + $250 = $ 600

After applying the Inter Spread Credit to each leg of the spreads formed between the correlated commodity, there is a total savings of ($350*0.55) + ($250 *0.55) = $330.

This total savings needs to be subtracted from the outright margin amount in order to get the final Inter Commodity Spread Charge on the portfolio.

Therefore, the final margin would be (Scanning Risk + Inter Month Spread Credit) – Cross Commodity Spread Charge.

The operation of the Cross Commodity Spread Module can be described in the Block Diagram as follows –



4. Problem Formulation

The SPAN Methodology to calculate the Initial Margin of the portfolio containing combined commodities is an extensive algorithm. Calculating the Initial Margin on a portfolio of a few 100 orders can be computed utilizing the modern computational tools. The SPAN Algorithm is used by the exchanges a few times a day to check on client accounts to ensure compliance with the current Initial Margin Requirements. The Initial Margin is calculated on accounts in which the orders are matched with the buyer. On the basis of the complexity theory, a rough estimate of the complexity analysis is of the order of O(16N), where N being the number of orders in the account. The 16 in the Big O Notation, is due to the 16 scenario's calculated during the Scanning Risk Computation where the position losses are calculated for 16 possible scenarios to determine the worst price movement for the given current market data.

The exchange provides access to its members to the exchange data at a premium. The majority of the members of the exchange being the High Frequency Trading Firms (HFT Firms), there is a need to monitor the client accounts & ensure that they do not cause an undue increase in the Initial Margin Requirement set by the exchange on it. Hence, there is also a need to check whether the unmatched orders in the portfolios rather along with the matched orders. Hence, the computation becomes very complex when taking into consideration both the matched & unmatched orders keeping in mind the Initial Margin Requirements for the portfolio.

Currently, the High Frequency Trading Environments adjust their order books constantly in the scale of microseconds, and strongly assert the need for real time computation of the pre-trade risk checks.

5 Implementation

5.1 Software Implementation of the SPAN Algorithm

To get a better understanding of the SPAN Algorithm, it was first implemented in C++, as it has been described in the previous sections.

For simulating the SPAN Algorithm on a portfolio, a large number of orders had to be generated quickly to check the accuracy of the algorithm. So for this purpose, we had a script to generate different types of data for a portfolio. For the SPAN Algorithm the fixed parameters remain the same for all the simulations, whereas the portfolio data keeps on changing.

The SPAN Algorithm was tested in C++ on a variety of sample portfolios and for a related Risk Parameter Files.

The implementation of the Algorithm in C++ helped us heavily in the further SystemVerilog Implementation of the Algorithm.

The C++ implementation is given in the section on Source Code.

Many additional developments were made after the implementation of the C++ code, while writing the SystemVerilog code.

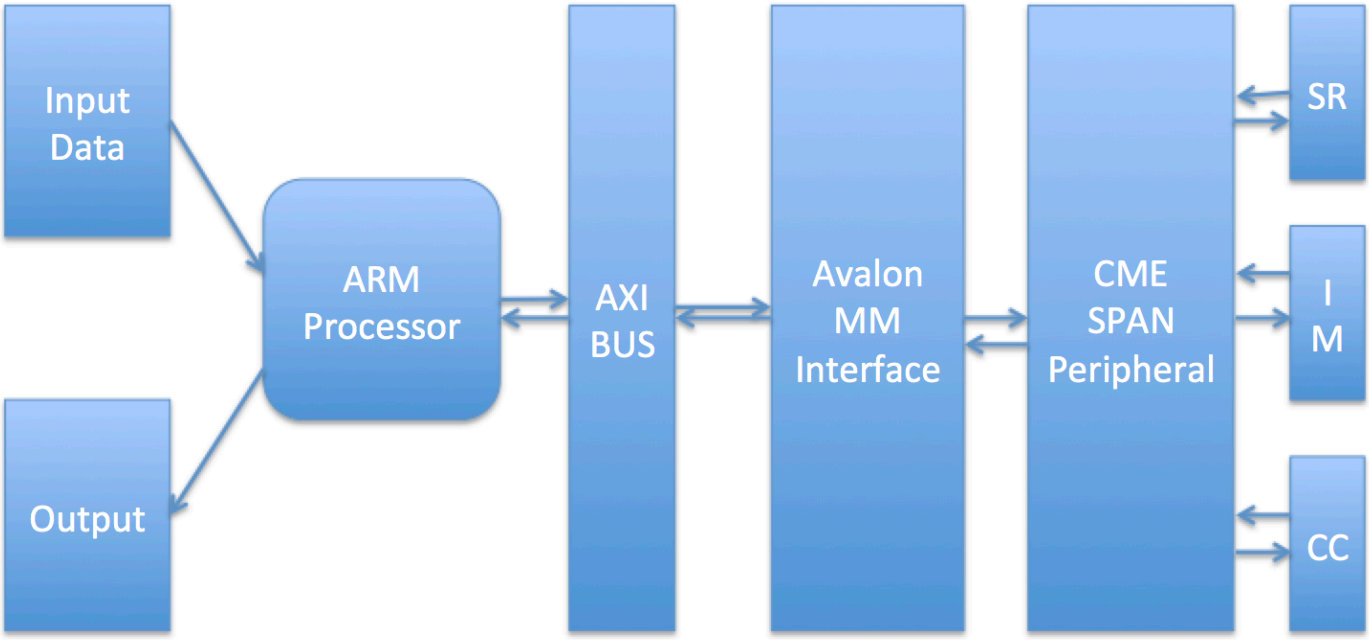
The outputs from the ModelSim and the SystemConsole were checked with the output of the C++ code and were found to match at each stage.

The C++ Code has been given in the Appendix and describe in detail the various operations with comments.

5.2 Hardware Implementation

The SystemVerilog implementation of the SPAN Algorithm was programmed on the FPGA board and was verified using the SystemConsole and the otuputs from the ARM processor.

One of the benefits of the Altera Cyclone V FGPA is its 925 MHz, dual-core ARM® Cortex™-A9 MPCore™ processor. The Altera SoCs integrate the ARM-based hard processor system (HPS) consisting of a processor, peripherals, and memory interfaces with the FPGA fabric using a high-bandwidth interconnect backbone.



Block Diagram

At this point we proceeded to build the software for the ARM core on the Altera Cyclone V FPGA. By improvising on the code in the Lab3 Tutorial, a kernel module was created to interface the FPGA fabric with the Avalon Memory Mapped Interface.

The Avalon MM bus functions by treating the peripheral components as a slave and the ARM core as a master. The SPAN peripherals are implemented such that it accepts data entries containing the portfolio data and the risk parameter files. The FPGA fabric after computation provides an output containing the three modules computed (Scanning Risk, InterMonth Spread Charge and the Cross Commodity Credit) and the final Initial Margin requirement. So the inputs are fed from to the CME SPAN peripheral from input text files and the output is obtained in the form of a output text file. The signals required for the Avalon MM interface to connect the ARM processor with the FPGA fabric are the*write, writedata, read, readdata, address, chipselect, clock*, and *reset.*

The Master (ARM core), initiates a write or a read to a certain address location. The Avalon MM bus on getting the request (read/write) and the address location decides on which peripheral (slave) to turn on and selects the peripheral using the chipselect.

The Avalon MM bus also sends a read/write depending on the Master's request and also a offset(address) signal which is decided by the address and the memory address the Master called to. The number of inputs also determined the number of offsets and the scope of the bits. If the number of inputs fed is less than the scope of the bits, there is wastage of resources. Also the burst functionality can be added, and will be implemented in the future work. The burst functionality requires additional signals like *burstcount, beginbursttransfer*, and *readdatavalid.*

With the availability of sufficient hardware resources, Initial Margin can be calculated for multiple portfolios in parallel and this would require the implementation of a lock signal so that only one core talks to a component at a time.

The SystemVerilog Implementation of the SPAN Algorithm, consists of three main modules.

1) Scanning Risk
2) InterMonth Spread Charge & Tier Spread Calculation(To calculate the Tier Spread Charge for the InterMonth Spread Credit)
3) Cross Commodity

The top level file – cme_span is a top level file for all the three modules.

5.2.1 Verification in ModelSim

The SystemVerilog Code of the SPAN Algorithm was verified in ModelSim. A Test Bench was constructed to calculate the Initial Margin for different portfolio data and risk parameter inputs.

For example considering the portfolio containing gold as an underlying asset:

Instrument	Future	Future	Future	Future	Future	Future
Position	15	-5	10	-15	5	5
Maturity	1	1	3	3	5	5

And the following as the inputs of the Risk Parameter File:

Price Scan Range – 96 $

Outright Rate – Gold 175 $, Silver 250 $

Ratio – 2:1

Inter Rate – 55 %.

Tier Spread Table:

Tier	Maturity	Tier 1	Tier 2	Tier 3
Tier 1	0-1 months	50 USD	X	X
Tier 2	1-2 months	80 USD	60 USD	X
Tier 3	2-3 months	90 USD	100 USD	70 USD

We get the values of the three parameters of the SPAN Algorithm as –

Scanning Risk – 480 $

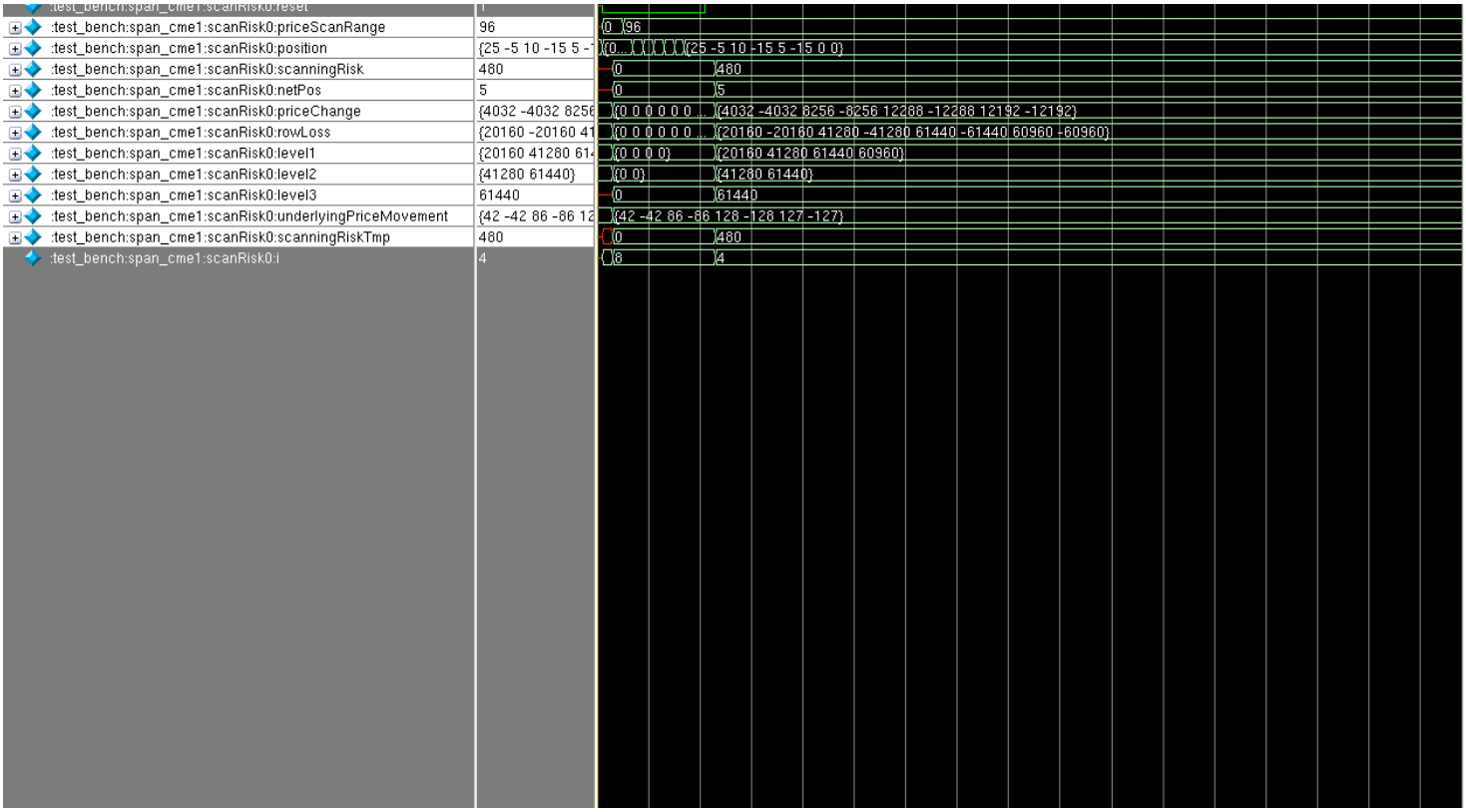
Inter Month Spread Charge – 2470 $

Cross Commodity Charge - 330 $

Initial Margin Requirement – 2620 $

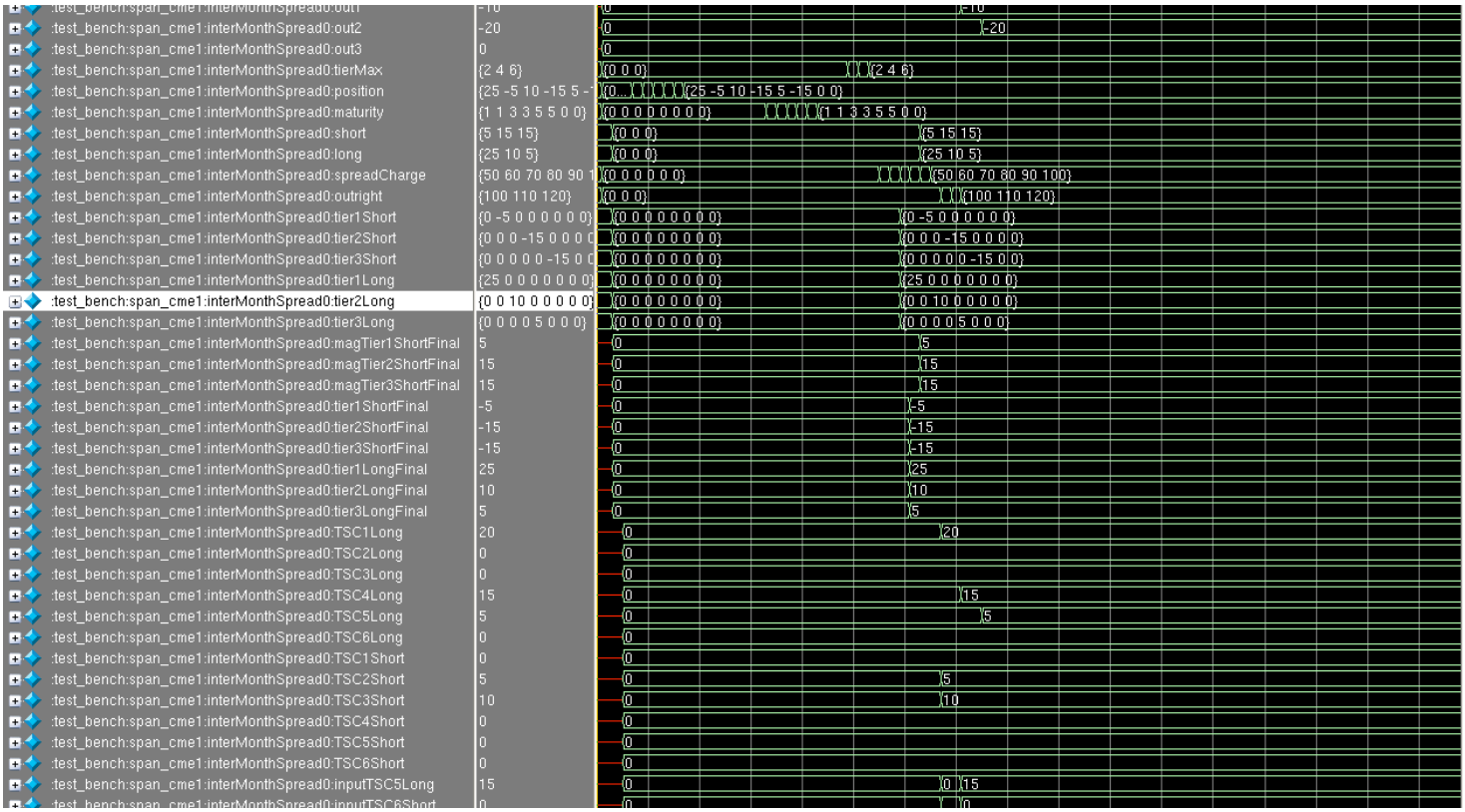
It can be very well seen that the Initial Margin Requirement is equal to (Scanning Risk + InterMonth Spread Charge – Cross Commodity Charge).

The Scanning Risk is verified in ModelSim and the output is observed as follows –



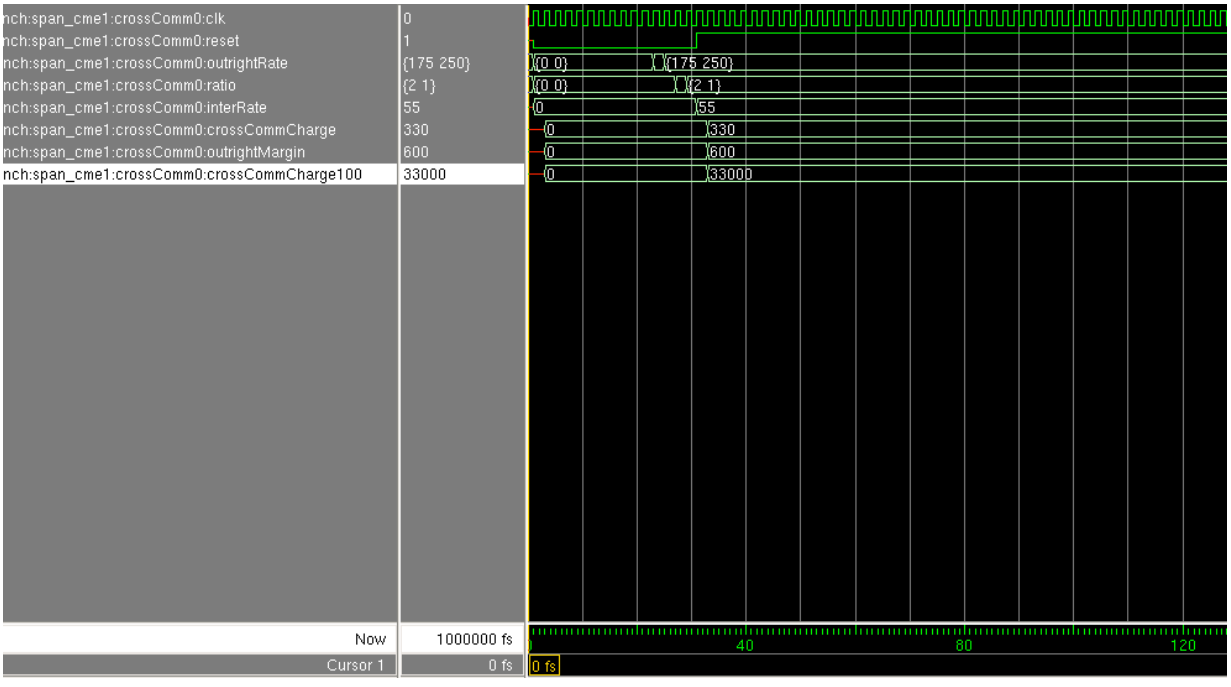
InterMonth Spread Charge –

The InterMonth Spread Charge module is verified using ModelSim and the value is verified and found to be –



Cross Commodity Credit

The Cross Commodity module of the Verilog is verified in ModelSim and the output is found to match the ones we got from the C++ model.



5.2.2 Implementing the Algorithm on the FPGA Board.

The SystemVerilog Implementation of the SPAN Algorithm was also burnt on the FPGA board and the output was verified with the outputs from the C++ code and the ModelSim Simulation.

a) SystemConsole Verification

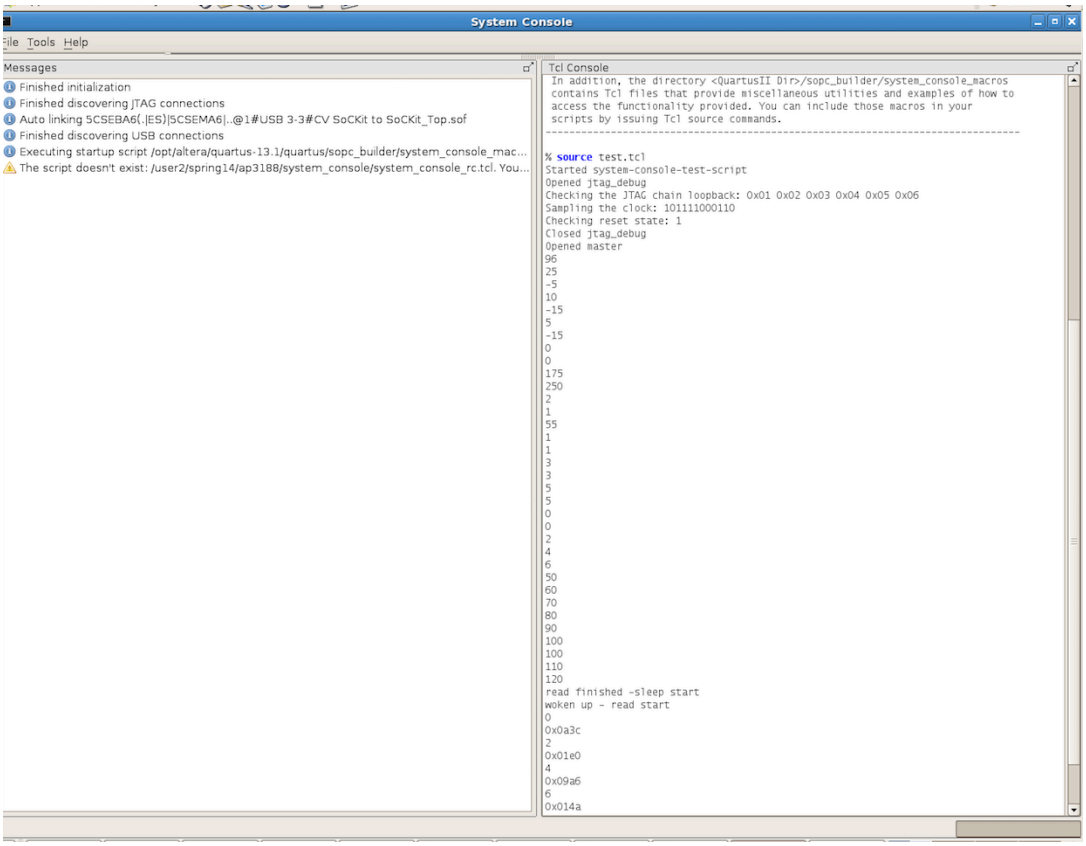
The System Verilog Code was burnt on the FPGA board and was tested using the SystemConsole. The output of the SystemConsole matched the outputs from the ModelSim and C++ implementation.

The outputs we get from the SystemVerilog Implementation are the

1) Scanning Risk
2) InterMonth Spread Charge
3) Cross Commodity Credit
4) Initial Margin Requirement

Getting the individual outputs of the modules, are beneficial as it would help to verify the final Initial Margin Output.

Figure below shows the output value of the Initial Margin Requirement of the SystemConsole is as follows



b) Using the ARM Processor

One of the main benefits of the Altera Cyclone V FPGA is the on board ARM Processor. The Arm Processor was used to give inputs from a Text File containing the Risk parameters and the Portfolio Data.

The output files containing the Initial Margin is obtained in the form of a text file.

The Initial Margin value was found to be the same as the one obtained from the C++ implementation and the ModelSim Verification.

The C code used to read from the input files and feed data to the ARM processor is also given in the Source Code Section.

The SystemVerilog Implementation was also checked to calculate the Initial Margin requirements of many different types of portfolios.

6 Results

The CME SPAN Algorithm was implemented in C++ and SystemVerilog and its functionality was checked using ModelSim and on the FPGA board. The results of the simulation in ModelSim are described and found to match the results from the FPGA fabric – from SystemConsole and the ARM Processor.

The project was concentrated for the Futures Market and involved a lot of optimization in the code for the proper functioning and taking into consideration all the corner cases of the SPAN Algorithm.

The implementation of the CME SPAN Algorithm was tested on various different types of test cases and was verified for its functionality for robustness. The implementation was found to function properly for all single and multiple portfolios.

7 Conclusion

The project was a great learning experience as it involved a great exposure to the Risk Management Systems in the High Frequency Trading Environments. The project also made us aware about the designing of Hardware for the Trading Industry and important and through the design should be. With the current implementation of CME SPAN Algorithm for the Futures Market , we believe that by working on it over the coming semester, we can improvise the implementation to take into account the options contracts in the portfolio and make the project a full-fledged implementation of CME- SPAN.

The implementation of the CME-SPAN for options would require the use of Black-Scholes Model or the Jump Diffusion Models for the options pricing and would require the complex computation and rigorous programming in SystemVerilog.

The goals for the future work would be –

1) Implementing the Algorithm to take into account the options contracts in the portfolio.

2) Short Option Minimum module to be included in the SPAN Algorithm for the Options Market.

3) Integrate the SPAN implementation with the current pre-trade risk checking infrastructure.

8 References

1) http://www.cmegroup.com/clearing/files/span-methodology.pdf

2) http://www.math.kth.se/matstat/seminarier/reports/M-exjobb12/120807b.pdf

3) http://www.cmegroup.com/clearing/margins/spread-calc.html

4) http://www.cftc.gov/files/tm/tmspan_margining043001.pdf

9 Special Thanks to

Prof. David Lariviere

Prof. Stephen Edwards

Qiushi Ding – TA

10 Appendix

a) C++ Implementation of the Code

b) C++ based model for CME SPAN

```
c) #include<iostream>
d) #include<conio.h>
e) #include<math.h>
f) #include<ctime>
g) #include<time.h>
h) #include<winsock.h>
i)
j) using namespace std;
k) #define PATH "C:/ReadMe.txt"
l) #define PATHPA "C:/RiskArray.txt"
m) #define PATHPort "C:/Portfolio.txt"
n) #define PATHRAr "C:/ParameterFile.txt"
o) #define PATHTDT "C:/TierDivisionTable.txt"
p) #define PATHTST "C:/TierSpreadTable.txt"
q) #define PATHDST "C:/DeltaSpreadTable.txt"
r) #define PATHSPT "C:/SpreadPrioTable.txt"
s) #define PATHOC "C:/OutrightCharge.txt"
t) #define PATHRatio "C:/Ratio.txt"
u) #define PATHInterRate "C:/InterCharge.txt"
v)
w) //ASSUMPTIONS
x) #define Spread_Charge 25   //defining charges acc to table 1.13 of KTH
   thesis paper
y) #define Outright_Charge 50
z) #define CuNetPos -15
aa)     #define CuWFPR 70
bb)     #define Credit 0.4
cc)
dd)     void main()
ee)     {
ff)          FILE * readPort, *readRAr, *readPA, *readTDT, *readTST, *readDST,
   *readSPT, *readOC, *readRatio, *readInterRate;
gg)          int netposdel = 0, DMC = 0;
hh)          SYSTEMTIME start, end, diff;
ii)          readRAr = NULL;
jj)          readPort = NULL;
kk)          readPA = NULL;
ll)          int pos, mat, price, th, vol, quantile, psr, volsr, intRate,
   VC[16], W[16];
```

```
mm)          float PSR, posLoss[16], PC[16], Long[12], Short[12];
nn)          char asset[20];
oo)          GetSystemTime(&start);
pp)          fopen_s(&readPort, PATHPort, "r");    //Reading Portfolio data
qq)          fopen_s(&readPA, PATHPA, "r");   //Reading Parameter Array
rr)          fopen_s(&readRAr, PATHRAr, "r"); //Reading Risk Array data
ss)          fopen_s(&readTDT, PATHTDT, "r"); //Reading Tier Division table
tt)          fopen_s(&readTST, PATHTST, "r"); //Reading Tier Spread table data
uu)          fopen_s(&readDST, PATHDST, "r"); //Reading Delta spread table
   data
vv)          fopen_s(&readSPT, PATHSPT, "r"); //Reading Spread Priority table
   data
ww)          fopen_s(&readOC, PATHOC, "r");        //Reading Outright charge
   data
xx)
yy)          //related to outright chare for cross-commodity
zz)          fopen_s(&readRatio, PATHRatio, "r"); //reaing Rato for differnt
   commodity - cross commodity
aaa)         fopen_s(&readInterRate, PATHInterRate, "r");     //related to
   inter charge or weightage for cross-commodity
bbb)
ccc)         for (int i = 0; i < 16; i++)
ddd)         {
eee)                                                          posLoss[i] = 0;
fff)              Long[i] = 0;
ggg)              Short[i] = 0;
hhh)         }          //initialising posLoss Array
iii)
jjj)
kkk)         if ((readPort != NULL) && (readRAr != NULL) && (readPA != NULL))
        //Read if Portfolio file opened
lll)         {
mmm)              fscanf_s(readPort, "%d\t%d\t%d", &pos, &mat, &price);
        //read 1st line of portfolio file
nnn)              fscanf_s(readRAr, "%d %d %d %d %d %d\n", &vol, &th,
   &quantile, &psr, &volsr, &intRate);   //read First line of Parameter file
ooo)              PSR = price * (vol/100.0) * (sqrt(float(th) / 252.0)) *
   quantile;
ppp)    //PSR(Price Scan Range) calculated each time Portfolio file is read
qqq)
rrr)    for (int i = 0; i < 16; i++)
sss)              {
ttt)                   if (readPA)
uuu)                        fscanf_s(readPA, "%f\t%d\t%d\n", &PC[i], &VC[i],
   &W[i]);
vvv)                   else
www)                        std::cout << "Invalid number of entires in Risk
   Parameter File" << endl;
xxx)
yyy)                        posLoss[i] += (PC[i] * float(W[i] / 100.0) * PSR)
   * pos;     //Position loss for each row of Risk array
zzz)              }
aaaa)
bbbb)             int month = (mat / 31), DMSC = 0, DMOC = 0; //month charge
   declarations
cccc)             netposdel += pos;
dddd)             if (pos < 0)
eeee)             {
ffff)                  Short[month] += pos;
```

```
gggg)               }
hhhh)               else
iiii)               {
jjjj)                    Long[month] += pos;
kkkk)               }
llll)
mmmm)               int j = 1, maxMonth = month; //j meant for index of mat and
   pos, maxMotnth stores the max month reached
nnnn)               while(!feof(readPort))
oooo)               {
pppp)                    fscanf_s(readPort, "%d\t%d\t%d\n", &pos, &mat,
   &price);
qqqq)                    j++;
rrrr)                    netposdel += pos;
ssss)                    month = mat / 31;     //get month value after each
   reada of mat
tttt)                    PSR = price * (vol / 100.0) * (sqrt(float(th) /
   252.0)) * quantile;
uuuu)                    for (int i = 0; i < 16; i++)
vvvv)                    {
wwww)                         posLoss[i] += (PC[i] * (W[i] / 100) * PSR) * pos;
        //Calculating position loss
xxxx)                    }
yyyy)                    //making Long and Short arrays
zzzz)                    if(pos < 0)
aaaaa)                   {Short[month] += pos; }
bbbbb)                   else
ccccc)                   {Long[month] += pos; }
ddddd)                   maxMonth = (month > maxMonth) ? month : maxMonth;
eeeee)                   if (Long[month] > -(Short[month]))
fffff)  //as soon as a long and short exist for the same month, make short 0
   if possible
ggggg)                   {
hhhhh)                        Long[month] += Short[month];
iiiii)                        DMSC += Short[month] * Spread_Charge; //add to
   DMSC everytime there is a non zero short
jjjjj)                        Short[month] = 0;
kkkkk)                   }
lllll)                   else
mmmmm)                   {
nnnnn)                        Long[month] = 0;
ooooo)                        DMSC += -(Long[month] * Spread_Charge);
ppppp)                        Short[month] += Long[month];
qqqqq)                   }
rrrrr)              }//end of while
sssss)              int check[12];
ttttt)              for (int z = 0; z < 12; z++)
uuuuu)              {
vvvvv)                   check[z] = 0;
wwwww)              }
xxxxx)
yyyyy)              for (int k = 0; k <= maxMonth; k++)   //handle intermonth
   spread
zzzzz)              {
aaaaaa)                  if (Short[k] != 0)
bbbbbb)                  {
cccccc)                       for (int l = 0; l < k; l++)
dddddd)                       {
eeeeee)                            check[l] = 0;
```

```
ffffff)                              if ((Long[l] > -(Short[k])) && (Short[k] !=
   0))
gggggg) //as soon as a long and short exist for the same month make short 0 if
   possible
hhhhhh)                              {
iiiiii)                                   Long[l] += Short[k];
jjjjjj)                                   DMSC += Short[k] * Spread_Charge;
        //add to DMSC each time there is a non-zero short
kkkkkk)                                   Short[k] = 0;
llllll)                                   check[l] = 1;
mmmmmm)                              }
nnnnnn)                              else
oooooo)                              {
pppppp)                                   Long[l] = 0;
qqqqqq)                                   DMSC += -(Long[l] * Spread_Charge);
rrrrrr)                                   Short[k] += Long[l];
ssssss)                              }
tttttt)                         }
uuuuuu)                    }
vvvvvv)               }//end of for
wwwwww)
xxxxxx)               for (int k = 0; k <= maxMonth; k++)   //find Delivery Month
   outright charge(DMOC)
yyyyyy)               {
zzzzzz)                    if (check[k] == 1)
aaaaaaa)                        DMOC += Long[k] * Outright_Charge;
bbbbbbb)              }
ccccccc)              DMC = DMOC - DMSC;    //DMSC is -ve so subtract to add abs
   values - Delivery Month Charge
ddddddd)
eeeeeee)         }//end of topmost if
fffffff)         float maxPosLoss = posLoss[0], pairedPosLoss = 0, VASC, TimeRisk
   = 0, wFutPrRisk, ICSC = 0;
ggggggg)//declarations reqd for section 4
hhhhhhh)
iiiiiii)for (int i = 0; i < 16; i++)
jjjjjjj)         {
kkkkkkk)               if (maxPosLoss > posLoss[i])
lllllll)               {
mmmmmmm)                    maxPosLoss = posLoss[i];
nnnnnnn)                    if ((i % 2) == 0)
ooooooo)                    {
ppppppp)                         pairedPosLoss = posLoss[i + 1];
qqqqqqq)                    }
rrrrrrr)                    else
sssssss)                    {
ttttttt)                         pairedPosLoss = posLoss[i - 1];
uuuuuuu)                    }
vvvvvvv)               }
wwwwwww)
xxxxxxx)         }
yyyyyyy)
zzzzzzz)         //start of calc for intercommodity spread credit
aaaaaaaa)
bbbbbbbb)              VASC = -(maxPosLoss + pairedPosLoss) / 2;
cccccccc)              TimeRisk = (posLoss[0] + posLoss[1]) / 2;
dddddddd)              wFutPrRisk = (VASC - TimeRisk) / netposdel;
eeeeeeee)              int PCS1 = Credit * -(CuNetPos) * wFutPrRisk;
ffffffff)              int PCS2 = Credit * -(CuNetPos)* CuWFPR;
```

```
ggggggg)          ICSC = PCS1 + PCS2;
hhhhhhh)
iiiiiiii)         //end of calc for section 4
jjjjjjjj)
kkkkkkk)          //second part of calculations
lllllllll)        int tier[10], T, SpArray, count = 0, TSpreadArr[40],
   DSpreadArr[30], j = 0;
mmmmmmmm)         while (!feof(readTDT))
nnnnnnnn)         {
ooooooo)               count++;
ppppppp)               fscanf_s(readTDT, "%d\t", &T);
qqqqqqqq)              fscanf_s(readTDT, "%d\n", &tier[T - 1]);
rrrrrrrr)         }
ssssssss)         while (!feof(readTST))
ttttttttt)        {
uuuuuuuu)              int i, temp;
vvvvvvvv)              fscanf_s(readTST, "%d\t", &temp);
wwwwwwww)              for (i = 0; i < count - 1; i++)
xxxxxxxx)                   fscanf_s(readTST, "%d\t", &TSpreadArr[j++]);
        //read as 50,0,0,80,60,0,90,100,70
yyyyyyyy)              fscanf_s(readTST, "%d\n", &TSpreadArr[j++]);
zzzzzzzz)         }
aaaaaaaaa)        while (!feof(readDST))
bbbbbbbbb)        {
ccccccccc)             int i = 0, temp;
ddddddddd)             for (int i = 0; i < count; i++)
eeeeeeeee)             {
fffffffff)                  fscanf_s(readDST, "%d\t%d\t%d\n", &temp,
   &DSpreadArr[(2 * i)], &DSpreadArr[(2 * i) + 1]);
ggggggggg)    //store long0, short0, long1, short1..
hhhhhhhhh)             }
iiiiiiiii)        }
jjjjjjjjj)        int TSC = 0;
kkkkkkkkk)        while (!feof(readSPT))
lllllllll)        {
mmmmmmmmm)             char col, row, ind;
nnnnnnnnn)             fscanf_s(readSPT, "%d %d\n", &col, &row);
ooooooooo)             ind = (3 * (row - 1)) + col - 1 ;     //index for
   TSpreadArr
ppppppppp)             col = (col - 1) * 2;
qqqqqqqqq)             row = (2 * row) - 1;  //convert row and col to
   corresponding indices in DSpreadArr
rrrrrrrrr)             if ((DSpreadArr[col] != 0) && (DSpreadArr[row] != 0))
sssssssss)    //For example, 1 to 3 will access 15 and -5 from DSpreadArr and
   90 from TSpreadArr
ttttttttt)
uuuuuuuuu)    {
vvvvvvvvv)                  TSC += DSpreadArr[row] * TSpreadArr[ind]; // calc
   TSC and Long - Short
wwwwwwwww)                  DSpreadArr[col] = (DSpreadArr[col] +
   DSpreadArr[row] >= 0) ? (DSpreadArr[col] + DSpreadArr[row]) : 0;
xxxxxxxxx)    //assign diff only if Long > Short
yyyyyyyyy)                  DSpreadArr[row] = (DSpreadArr[col] +
   DSpreadArr[row] >= 0) ? 0 : (DSpreadArr[row] + DSpreadArr[col]);
zzzzzzzzz)    //assign diff only if Long > Short
aaaaaaaaaa)             }
bbbbbbbbbb)       }     //end of if
ccccccccccc)
ddddddddddd)
```

```
eeeeeeeeee)        //Beginning of Cross-commodity calculations
fffffffff)         int CrossOutCharge, CrossRatio, CrossComCharge = 0;
ggggggggg)         int intrRate;
hhhhhhhhh)         int i = 0;
iiiiiiiii)         if (!(fscanf_s(readInterRate, "%d", &intrRate)))
jjjjjjjjj)         {
kkkkkkkkk)              std::cout << "FILE READ ERROR" << endl;
lllllllll)         }
mmmmmmmmm)         while (!feof(readOC))
nnnnnnnnn)         {
ooooooooo)              fscanf_s(readOC, "%d ", &CrossOutCharge);
ppppppppp)              fscanf_s(readRatio, "%d ", &CrossRatio);
qqqqqqqqq)              CrossComCharge += CrossRatio * CrossOutCharge;
rrrrrrrrr)         }
sssssssss)
ttttttttt)         CrossComCharge = (float)CrossComCharge * (float)intrRate /
   100.0;
uuuuuuuuu)
vvvvvvvvv)         //std::cout << "\n  CrossCharge : " << CrossComCharge <<
   endl;
wwwwwwwww)         GetSystemTime(&end);
xxxxxxxxx)         //closing all opened files
yyyyyyyyy)         float InitMargReq = (-maxPosLoss) + (-TSC) + DMC - ICSC -
   CrossComCharge; //Intitial Margin Requirement
zzzzzzzzz)
aaaaaaaaa)         std::cout << endl << InitMargReq << endl;
bbbbbbbbb)
ccccccccc)
ddddddddd)         //closing all opened text files
eeeeeeeeee)        if (readPA)
fffffffff)         {
ggggggggg)              fclose(readPA);
hhhhhhhhh)         }
iiiiiiiiii)        if (readPort)
jjjjjjjjj)         {
kkkkkkkkk)              fclose(readPort);
lllllllll)         }
mmmmmmmmm)         if (readRAr)
nnnnnnnnn)         {
ooooooooo)              fclose(readRAr);
ppppppppp)         }
qqqqqqqqq)         if (readTST)
rrrrrrrrr)         {
sssssssss)              fclose(readTST);
ttttttttt)         }
uuuuuuuuu)         if (readDST)
vvvvvvvvv)         {
wwwwwwwww)              fclose(readDST);
xxxxxxxxx)         }
yyyyyyyyy)         if (readSPT)
zzzzzzzzz)         {
aaaaaaaaa)              fclose(readSPT);
bbbbbbbbb)         }
ccccccccc)         if (readTDT)
ddddddddd)         {
eeeeeeeeee)              fclose(readTDT);
fffffffff)         }
ggggggggg)         if (readOC)
hhhhhhhhh)         {
```

```
iiiiiiiiiiii)            fclose(readOC);
jjjjjjjjjjjj)       }
kkkkkkkkkkk)        if (readRatio)
llllllllllll)       {
mmmmmmmmmmmm)            fclose(readRatio);
nnnnnnnnnnn)        }
ooooooooooo)        if (readInterRate)
pppppppppppp)       {
qqqqqqqqqqqq)            fclose(readInterRate);
rrrrrrrrrrr)        }
sssssssssss)
ttttttttttt)        diff.wMinute = start.wMinute - end.wMinute;
uuuuuuuuuuu)        diff.wSecond = start.wSecond - end.wSecond;
vvvvvvvvvvv)        diff.wMilliseconds = start.wMilliseconds -
   end.wMilliseconds;    //Total program run time calculations
wwwwwwwwwww)        std::cout << "\nTime Taken to complete program " <<
   (end.wMilliseconds - start.wMilliseconds) << "milliseconds " << endl;
xxxxxxxxxxx)        _getch();
yyyyyyyyyyy)  }
```

b)Verilog files

span_cme.sv

This is our top level file. It calls the scanningRisk.sv, interMonthSpread.sv and crossCommodityCharge.sv . When these sub-modules finish their calculation, they send these values back to the top module. The top module then combines all these values and sends the final answer and sub-module outputs to the software part. This module also keeps a count of the number of cycles that were taken to get the final output and reports that too.

It receive the following inputs:

clk

reset

writeData

offset

write

chipselect

It sends out the following outputs:

read

readData

The software part sends out the data from input file serially in writeData with the appropriate value of offset.

As the data arrives, the start signal to the sub-modules is sent.

When the done signal arrives from interMonthSpread sub-module (this module finishes calculation the last), all the values calculated by the sub-modules are combined and sent back to the software part.

```
module span_cme(

   input      logic                clk,

   input      logic                reset,

   input      logic [15:0]    writeData,

   input      logic [5:0]     offset,

   input      logic                write,

   input      logic                chipselect,

   input      logic                read,

   output     logic [15:0]    readData

);




/*************************************************************************
**********************************/

/*********************************************    Declarations
   ***************************************/

/*************************************************************************
**********************************/




   reg [15:0]      initialMargin;             //Final calculated value of
Initial Margin

   reg [15:0]      scanningRisk;              //First    Component:    The
Scanning Risk

   reg [15:0]      TSC;                            //Second    Component:
Tier Spread Charge

   reg [15:0]      crossCommCharge;      //Third     Component:     Cross
Commodity Charge
```

```
//Portfolio inputs

//Maximum number of instruments is 8

//Maximum number of Tiers is 3

   reg [15:0] priceScanRange;       //Price Scan Range

   reg [15:0]      position [0:7];       //Array    for    positions    of
instrument

   reg [7:0]  maturity [0:7];       //Array for maturity of instrument, in
months

   reg [3:0]  tierMax[0:2];              //Array   for   upper   limit   of
tiers, in months

   reg [7:0]  spreadCharge [0:5];   //Array   for   spread   charge   between
tiers, in order of priority

   reg [7:0]  outright [0:2];       //Array for outright rate for spreads
between tiers, in order of priority

   reg [15:0]      outrightRate[0:1];    //Array  for  Outright  rate  for
Cross Commodity Charge

   reg [7:0]  ratio[0:1];                     //Array for Ratio between 2
commodities for Cross Commodity Charge

   reg [7:0]  interRate;                 //Rate   for   Cross   Commodity
Charge




//Start signals for the 3 components

   logic           startScanRisk,             //Start    Scanning    Risk
Calculation

                        startInterMonth,      //Start  Intermonth  Spread
Charge calculation

                        startCross;                     //Start    Cross
Commodity Calculation



//Tier  Spread  Calculation  done,  representing  the  end  of  all  the  3
components.
```

```
   wire            spreadDone;


//Couter for keeping a track of number of cycles taken for the entire
calculation

   reg             startCount;

   reg [15:0] cyclesTaken;



//Loop index

   integer i;







   /*****************************************************************
**********************************/

   /*************************************************BODY
   ************************************************/

   /*****************************************************************
**********************************/



//Module for calculating the Scanning Risk

   scanRisk scanRisk0(.reset(startScanRisk), .*);



//Module for calculating Cross Commodity Charge

   crossComm crossComm0(.reset(startCross), .*);



//Module for calculating Inter Month Spread Charge

   interMonthSpread             interMonthSpread0(.reset(startInterMonth),
.done(spreadDone), .*);
```

```
   always_ff @ (posedge clk) begin


         if (reset) begin
                              //Preparing   for   calculation,   resetting
everything

                     startScanRisk         <= 1'd0;

                     startInterMonth  <= 1'd0;

                     startCross                  <= 1'd0;


                     priceScanRange   <= 16'd0;

                     interRate             <= 8'd0;


                     startCount            = 1'd0;
                               //Disabling the counter

                     cyclesTaken                 <= 16'd0;
                               //Resetting the counter




                     for (i = 0; i < 8; i = i + 1) begin

                           position[i] <= 16'd0;

                           maturity[i] <= 8'd0;

                     end




                     for (i = 0; i < 3; i = i + 1) begin

                           tierMax[i]      <= 4'd0;

                           outright[i] <= 8'd0;

                     end
```

```
                    for (i = 0; i < 6; i = i + 1) begin
                         spreadCharge[i] <= 8'd0;
                    end




                    for (i = 0; i < 2; i = i + 1) begin
                         outrightRate[i] <= 16'd0;
                         ratio[i]              <= 8'd0;
                    end



          end else if (chipselect && write) begin
               //Reading input data from C code


                    case (offset)
                         6'd0  :  begin


                                                  priceScanRange        <=
writeData[15:0];   //Price scan range

                                                  startCount
   = 1'd1;                          //Enabling   the   Counter   to   start
counting the number of cycles


                                             end

                         6'd1  :    position[0]                         <=
writeData[15:0];   //Positions of 1st instrument

                         6'd2  :    position[1]                         <=
writeData[15:0];   //Positions of 2nd instrument
```

```
                        6'd3  :    position[2]                    <=
writeData[15:0];   //Positions of 3rd instrument

                        6'd4  :    position[3]                    <=
writeData[15:0];   //Positions of 4th instrument

                        6'd5  :    position[4]                    <=
writeData[15:0];   //Positions of 5th instrument

                        6'd6  :    position[5]                    <=
writeData[15:0];   //Positions of 6th instrument

                        6'd7  :    position[6]                    <=
writeData[15:0];   //Positions of 7th instrument

                        6'd8  :    begin position[7]          <=
writeData[15:0];   //Positions of 8th instrument



                                              startScanRisk
   <= 1'd1;                    //Starting Scanning Risk calculation


                                        end

                        6'd9  :  outrightRate[0]              <=
writeData[15:0];   //Outright rate for 1st commodity

                        6'd10 :    outrightRate[1]            <=
writeData[15:0];   //Outright rate for 2nd commodity



                        6'd11 :    ratio[0]                       <=
writeData[7:0];    //Cross commodity ratio for 1st commodity

                        6'd12 :    ratio[1]                       <=
writeData[7:0];    //Cross commodity ratio for 2nd commodity

                        6'd13 :    begin interRate            <=
writeData[7:0];    //Cross commodity correlation factor



                                                 startCross
   <= 1'd1;                        //Starting Cross Commodity calculation


                                        end

                        6'd14 :    maturity[0]                    <=
writeData[7:0];    //Maturity for 1st instrument in months
```

```
                        6'd15 :    maturity[1]                    <=
writeData[7:0];    //Maturity for 2nd instrument in months

                        6'd16 :    maturity[2]                    <=
writeData[7:0];    //Maturity for 3rd instrument in months

                        6'd17 :    maturity[3]                    <=
writeData[7:0];    //Maturity for 4th instrument in months

                        6'd18 :    maturity[4]                    <=
writeData[7:0];    //Maturity for 5th instrument in months

                        6'd19 :    maturity[5]                    <=
writeData[7:0];    //Maturity for 6th instrument in months

                        6'd20 :    maturity[6]                    <=
writeData[7:0];    //Maturity for 7th instrument in months

                        6'd21 :    maturity[7]                    <=
writeData[7:0];    //Maturity for 8th instrument in months



                        6'd22 :    tierMax[0]
   <= writeData[3:0];    //Upper value of a Tier 1 in months

                        6'd23 :    tierMax[1]
   <= writeData[3:0];    //Upper value of a Tier 2 in months

                        6'd24 :    tierMax[2]
   <= writeData[3:0];    //Upper value of a Tier 3 in months



                        6'd25 :    spreadCharge[0]            <=
writeData[7:0];    //Charge for spread between tier 1 Long and tier 1
Short

                        6'd26 :    begin spreadCharge[1] <=
writeData[7:0];    //Charge for spread between tier 2 Long and tier 2
Short



                                                  startInterMonth <=
1'd1;                        //Early Start to Intermonth Spread Charge



                                       end

                        6'd27 :    spreadCharge[2]            <=
writeData[7:0];    //Charge for spread between tier 3 Long and tier 3
Short
```

```
                        6'd28 :    spreadCharge[3]            <=
writeData[7:0];    //Charge for spread between tier 1 Long and tier 2
Short

                        6'd29 :    spreadCharge[4]            <=
writeData[7:0];    //Charge for spread between tier 1 Long and tier 3
Short

                        6'd30 :    spreadCharge[5]            <=
writeData[7:0];    //Charge for spread between tier 2 Long and tier 3
Short



                        6'd31 :    outright[0]                     <=
writeData[7:0];    //Outright rate for spread between tier 1 Long and
tier 2 Short

                        6'd32 :    outright[1]                     <=
writeData[7:0];    //Outright rate for spread between tier 1 Long and
tier 3 Short

                        6'd33 :    outright[2]                     <=
writeData[7:0];    //Outright rate for spread between tier 2 Long and
tier 3 Short



                        default: begin  startScanRisk        <= 1'd0;

                                                    startInterMonth <=
1'd0;

                                                    startCross
   <= 1'd0;

                                        end

                  endcase



        end else if (chipselect && read ) begin
             //Passing the calculated margin to Readdata output

             case (offset)

                  6'd0: begin

                                  readData[15:0] <= initialMargin[15:0];
             //Sending out Initial Margin to software

                                  startCount      = 1'd0;
                             //Disabling the cycle counter

                             end
```

```
                    6'd1: readData[15:0]  <= scanningRisk[15:0];
          //Sending out Scanning Risk to software

                    6'd2: readData[15:0]  <= TSC[15:0];
               //Sending out Initial Margin to software

                    6'd3: readData[15:0]  <= crossCommCharge[15:0];
   //Sending out Cross Commodity Charge to software

                    6'd4: readData[15:0]  <= cyclesTaken[15:0];
   //Sending out the cycles counted

                    default:readData[15:0]     <= 16'd0;

               endcase



          end


//Counter that counts the number of cycles starting from arrival of data
till the output is sent on the Avalon Bus

          if (startCount)

               cyclesTaken <= cyclesTaken + 16'd1;

   end




   always_comb begin


          initialMargin   =   (spreadDone)   ?   (scanningRisk   +   TSC   -
crossCommCharge) : 16'd0;          //Final total of all the 3 components



   end




endmodule
```

scanningRisk.sv

Once this module receives the start signal from the top module it does the following.

It first calculates the sum of all the positions in the portfolio.

Depending upon the Price Scan Range value and the Risk parameters (hard coded) it determines the price change for all the risk scenarios.

It multiplies the price change with the net positions to calculate the loss as per each risk scenario.

It then selects the largest value out of all the possible losses and that is the output of this module (if the value is negative, then the output is 0. This module is calculating the worst case loss and a negative loss, which would mean a profit, is reset to 0).

```
`define compareMag(x,y) ((x)>(y)) ? (x) : (y);              //DEFINE:
Compares X and Y and returns the greater value

`define compareUnity(x,y,z) (x) ? (y) : (z);                //DEFINE: Checks
if X is unity and returns Y if true, else returns Z




module scanRisk(

 input      logic                clk,

 input      logic                reset,

 input      logic [15:0]    priceScanRange,

 input      logic [15:0]    position [0:7],

 output     logic [15:0]    scanningRisk

);







/*************************************************************************
*****************************/

/*********************************************  Declarations
 ***************************************/

/*************************************************************************
*****************************/



 logic [15:0]    netPos;
                              //Sum of all the positions in the portfolio

 logic [23:0]    priceChange[0:7];
                           //Change in the price as per the underlying
movement of the Price Scan Range for the 16 scenarios of Risk Array(only 8
scenarios are considered as volity change in not relavent for Futures)

 logic [31:0]    rowLoss[0:7];
                           //Loss for each of the 16(8) Risk Array scenarios

 logic [31:0]    level1[0:3];
                           //Array for selecting the greater value between 2
alternate Risk Array scenarios
```

```
 logic [31:0]    level2[0:1];
                           //Array for selecting the greater value between
the chosen(level1) values of Risk Array scenarios

 logic [31:0]    level3;
                                  //The greatest the Risk Array scenario

 logic [8:0]     underlyingPriceMovement[0:7];
                 //Risk Array underlying price movements

 logic [31:0]    scanningRiskTmp;
                      //Temporary  scanning  risk  value  to  check  if  it
negative or positive



//Loop index

 integer i;









 parameter       UNDERLYING33  = 8'd42,
                       //33% of 128, Wieght 100%

                            UNDERLYING67  = 8'd86,
                                 //67% of 128, Wieght 100%

                            UNDERLYING100 = 8'd128,
                                 //100% of 128, Wieght 100%

                            UNDERLYING300 = 8'd127;
                                 //300% of 128, Wieght 33%




/*************************************************************************
*****************************/

/********************************************** BODY
 **********************************************/

/*************************************************************************
*****************************/



 always_ff @ (posedge clk)  begin
```

```
      if(~reset) begin
                                    //Preparing   for   calculation,   resetting
everything
            scanningRisk = 16'd0;
            netPos = 16'd0;
            level3 = 32'd0;
            scanningRiskTmp = 16'd0;
            for (i = 0; i < 8; i = i + 1) begin
                  priceChange[i] = 24'd0;
                  rowLoss[i] = 32'd0;
            end
            for (i = 0; i < 4; i = i + 1) begin
                  level1[i] = 32'd0;
            end
            for (i = 0; i < 2; i = i + 1) begin
                  level2[i] = 32'd0;
            end
            for (i = 0; i < 8; i = i + 1) begin
                  case (i)
                        0 : underlyingPriceMovement[i] =  UNDERLYING33;
                        1 : underlyingPriceMovement[i] = -UNDERLYING33;
                        2 : underlyingPriceMovement[i] =  UNDERLYING67;
                        3 : underlyingPriceMovement[i] = -UNDERLYING67;
                        4 : underlyingPriceMovement[i] =  UNDERLYING100;
                        5 : underlyingPriceMovement[i] = -UNDERLYING100;
                        6 : underlyingPriceMovement[i] =  UNDERLYING300;
                        7 : underlyingPriceMovement[i] = -UNDERLYING300;
                  endcase
```

```
            end

      end else begin
                                        //Start Scanning Risk calculation


                                                  //Accumulating   all   the
positions

            netPos    =    (((position[0]+    position[1])+    (position[2]+
position[3]))+ ((position[4]+ position[5])+ (position[6]+ position[7])));




            for (i = 0; i < 4; i = i + 1) begin
                        //Underlying price movement of price scan range

                  priceChange[(2*i)]   =   underlyingPriceMovement[(2*i)]   *
priceScanRange;

                  priceChange[(2*i) + 1] = ~priceChange[(2*i)] + 1'd1;

            end




            for (i = 0; i < 4; i = i + 1) begin
                        //Row loss multiplied with price change, multiple of
128

                  rowLoss[(2*i)]  =  (netPos[15])  ?  (~(priceChange[(2*i)]  *
(~netPos + 1'd1)) + 1'd1): (priceChange[(2*i)] * netPos);

                  rowLoss[((2*i) + 1)] = ~rowLoss[(2*i)] + 1'd1;

            end




            if (netPos == 16'd0 || netPos[15])



                  scanningRisk = 16'd0;
                              //If net position is 0 then no need to perform
any comparison/multiplication
```

```
            else begin



                  for (i = 0; i < 4; i = i + 1) begin
                        //Level 1 comparison between rows with same value but
opposite signs, comparing sign bit with 1



                        level1[i]     =     `compareUnity(rowLoss[((2*i)     +
1)][31],rowLoss[(2*i)],rowLoss[((2*i) + 1)]);



                  end



            end



                                                      //Level     2     comparison
between (33% and 67%) and (100% and 300%) underlying price movements

            level2[0]  = `compareMag(level1[0],level1[1]);

            level2[1]  = `compareMag(level1[2],level1[3]);




            level3           = `compareMag(level2[0],level2[1]);
                  //Level 3 comparison between the winner of top 2




            scanningRiskTmp = level3 >> 7;
                        //Dividing by 128 for the final answer




            scanningRisk = scanningRiskTmp[15:0];
                  //If scanning risk is negetive then scanning risk in 0
```

```
      end


 end


endmodule
```

interMonthSpread.sv

This module receives the start signal from the top module even before the entire data required by the module is ready. We can achieve this because this module works in a step by step fashion. So it can be started early and by the time the algorithm arrives at the step which requires the last string of data, that data has arrived.

First the positions are sorted into tiers depending up on their maturity dates. After this they are accumulated to form the Tier Spread Table, which contains all the Long and Short positions in the different tiers.

Once the Tier Spread table is formed, the spread calculations begin. Calculation for spread 1, 2 and 3 is always started. These 3 happen in parallel. Once done, the updated Tier Spread Table is checked to see if other tiers (4, 5 and 6) can be formed. (These tier calculations are done in order, i.e. they have a priority.) If yes, then calculation for those spreads is triggered otherwise they are skipped.

After all the spreads have been calculated, they are added and sent back to the top module.

All the steps inside this module are controlled by an FSM.

```
module interMonthSpread(

 input      logic                clk,

 input      logic                reset,

 input      logic [3:0]     tierMax              [0:2],

 input      logic [15:0]    position        [0:7],

 input      logic [7:0]     maturity        [0:7],

 input      logic [7:0]     spreadCharge    [0:5],

 input      logic [7:0]     outright        [0:2],

 output     logic [15:0]    TSC,
```

```
 output     logic                 done

);
```

```
/*************************************************************************
*****************************/

/*****************************************  Declarations
 ************************************/

/*************************************************************************
*****************************/
```

```
 reg [15:0]      tier1Short[0:7],
      //Array of Shorts sorted by Maturity

                      tier2Short[0:7],

                      tier3Short[0:7],

                      tier1Long[0:7],
            //Array of Longs sorted my Maturity

                      tier2Long[0:7],

                      tier3Long[0:7];
```

```
 reg [6:0]  short[0:2],
            //Short's Tier spread table

                      long[0:2];
                 //Long's Tier spread table
```

```
 reg [15:0]      magTier1ShortFinal,
      //Magnitude of Shorts in Tier Spread table

                      magTier2ShortFinal,

                      magTier3ShortFinal;
```

```
reg [15:0] tier1ShortFinal,
//Total Short values in a Tier

                    tier2ShortFinal,

                    tier3ShortFinal,

                    tier1LongFinal,
          //Total Long values in a Tier

                    tier2LongFinal,

                    tier3LongFinal;


                         //Signals triggering spread calculation

reg            tsc123Start,
          //Triggering spread 1, 2 and 3 calculation

                    tsc4Start,
               //Triggering spread 4 calculation

                    tsc5Start,
               //Triggering spread 5 calculation

                    tsc6Start,
               //Triggering spread 6 calculation

                    tscDone,
                    //All spreads calculated

                    spreadTotalDone;
          //All spreads added up



reg [6:0]  TSC1Long,
     //Long results after Spread calculation

                    TSC2Long,

                    TSC3Long,

                    TSC4Long,

                    TSC5Long,

                    TSC6Long,

                    TSC1Short,
               //Short result for Spread calculation
```

```
                    TSC2Short,

                    TSC3Short,

                    TSC4Short,

                    TSC5Short,

                    TSC6Short;



reg [6:0]  inputTSC5Long,
     //Long input for Spread 5

                    inputTSC6Short;
          //Short input for Spread 6



reg [15:0]      TSC1,
                //Calculated Tier spreads

                    TSC2,

                    TSC3,

                    TSC4,

                    TSC5,

                    TSC6;



reg [15:0]      out1,
                //Outright charge of Spread 4

                    out2,
                    //Outright charge of Spread 5

                    out3;
                    //Outright charge of Spread 6




wire            positionsAccumulated,
//Signals informing the status about intermediate steps

                    spreadTableFormed,
```

```
                    goSpread4,
                //Signals informing whether Spreads 4, 5 and 6 are possible
or not

                    goSpread5,

                    goSpread6,

                    goSpread5From4,

                    goSpread6From4,

                    goSpread6From5;



 reg [3:0] state;
           //State variable



 localparam      IDLE            = 0,
           //Initial state

                    SORTPOS         = 1,
                //State  for  starting  the  sorting  of  Positions  based  on
Maturity

                    ACCPOS          = 2,
                //State for adding all the positions in a Tier

                    SPREAD123  = 3,
           //State for calculating spreads 1, 2 and 3

                    WAIT123         = 4,
                //State for waiting for the values of spreads 1, 2 amd 3 to
settle

                    SPREAD4         = 5,
                //State for calculating spread 4

                    WAIT4           = 6,
                //State for waiting for the values of spread 4 to settle

                    SPREAD5         = 7,
                //State for calculating spread 5

                    WAIT5           = 8,
                //State for waiting for the values of spread 5 to settle

                    SPREAD6         = 9,
                //State for calculating spread 6

                    SPREADTOTAL     = 10,
                //State for adding up all the spreads
```

```
                    DONE            = 11;
                //State representing that Inter Month Spread calculation is
Done



//Loop Index

 integer i;




/*************************************************************************
*****************************/

/********************************************** BODY
 *********************************************/

/*************************************************************************
*****************************/



                            //Signal  informing  that  positions  have  been
sorted and accumulated

 assign     positionsAccumulated =     ((tier1ShortFinal    !=    8'b0)    ||
(tier1LongFinal != 8'b0) || (tier2ShortFinal != 8'b0) || (tier2LongFinal !=
8'b0) || (tier3ShortFinal != 8'b0) || (tier3LongFinal != 8'b0)) ? 1'b1 :
1'b0;


                            //Signal  informing  that  spread  table  has  been
formed

 assign     spreadTableFormed     = ((long[0] != 0) || (short[0] != 0) ||
(long[1] != 0) || (short[1] != 0) || (long[2] != 0) || (short[2] != 0));




 assign     goSpread4                   = ((TSC1Long != 0) && (TSC2Short !=
0));                //Spread 4 can be formed

 assign     goSpread5                   = ((TSC1Long != 0) && (TSC3Short !=
0));                //Spread 5 can be formed

 assign     goSpread6                   = ((TSC2Long != 0) && (TSC3Short !=
0));                //Spread 6 can be formed

 assign     goSpread5From4        = ((TSC4Long != 0) && (TSC3Short != 0));
            //Spread 5 can be formed after spread 4
```

```
assign     goSpread6From4        = ((TSC2Long != 0) && (TSC3Short != 0));
           //Spread 6 can be formed after spread 4

assign     goSpread6From5        = ((TSC2Long != 0) && (TSC5Short != 0));
           //Spread 6 can be formed after spread 5



assign     done                            = spreadTotalDone;
                                     //Spreads formed









always_ff @(posedge clk) begin



      if (~reset) begin
                                                    //Initializing    FSM,
resetting everything

                                  tsc123Start                          <=
1'd0;

                                  tsc4Start                            <=
1'd0;

                                  tsc5Start                            <=
1'd0;

                                  tsc6Start                            <=
1'd0;

                                  tscDone
 <= 1'd0;

                                  spreadTotalDone                 <= 1'd0;

                                  state
<= IDLE;

      end else begin
                                                    //FSM starts



            case (state)
```

```
                IDLE: begin
                                                        //STATE 0

                                    tsc123Start                             <=
1'd0;

                                    tsc4Start                               <=
1'd0;

                                    tsc5Start                               <=
1'd0;

                                    tsc6Start                               <=
1'd0;

                                    tscDone
 <= 1'd0;

                                    spreadTotalDone                 <= 1'd0;

                                    state
 <= SORTPOS;                                            //Always  go  to
next state for sorting positions

                            end



                SORTPOS:   begin
                                                        //STATE 1

                                        state
 <= ACCPOS;                                             //Always  go  to  next
state for  accumulating all the positions

                                    end



                ACCPOS:    begin
                                                        //STATE 2



                                        if     ((positionsAccumulated)     &&
(spreadTableFormed)) begin

                                                    state
 <= SPREAD123;                                          //Start  spread  1,  2
and 3 calculation after positions have been sorted and spread table is formed

                                                    tsc123Start         <=
1'd1;
```

```
                                    end else state              <= ACCPOS;
                                    //Else stay in the same state


                                end


                SPREAD123: state                                        <=
WAIT123;                                        //STATE 3


                WAIT123:   begin
                                                //STATE 4


                                    if (goSpread4) begin
                                                //Start      spread      4
calculation (the next spread in sequence)

                                        state
 <= SPREAD4;

                                        tsc4Start                   <=
1'd1;


                                    end else if (~goSpread4 && goSpread5)
begin                           //Jump  to  spread  5  calculation  skipping
spread 4

                                        state
 <= SPREAD5;

                                        tsc5Start                   <=
1'd1;


                                    end else if (~goSpread4 && ~goSpread5
&& goSpread6) begin    //Jump to spread 6 calculation skipping spread 4 and 5

                                        state
 <= SPREAD6;

                                        tsc6Start                   <=
1'd1;
```

```
                                        end else if (~goSpread4 && ~goSpread5
&& ~goSpread6)begin    //No more spreads can be formed, add up spread 1, 2
and 3

                                              state
 <= SPREADTOTAL;

                                              tscDone
 <= 1'd1;



                                        end


                                  end


                SPREAD4: state                                  <= WAIT4;
                                  //STATE 5


                WAIT4:     begin
                                                      //STATE 6


                                        if (goSpread5From4)begin
                                             //Start   spread   5   calculation
(the next spread in sequence)

                                              state
 <= SPREAD5;

                                              tsc5Start                  <=
1'd1;



                                        end   else   if   (~goSpread5From4   &&
goSpread6From4) begin             //Jump  to  spread  6  calculation  skipping
spread 5

                                              state
 <= SPREAD6;

                                              tsc6Start                  <=
1'd1;
```

```
                                    end else if (~goSpread5From4 &&
~goSpread6From4) begin      //No more spreads can be formed, add up spread 1,
2, 3 and 4

                                        state
 <= SPREADTOTAL;

                                        tscDone
 = 1'd1;



                                    end


                                end


                SPREAD5: state                                  <= WAIT5;
                                    //STATE 7


                WAIT5: begin
                                                    //STATE 8


                                if (goSpread6From5) begin
                                        //Start  spread  6  calculation
(the next spread in sequence)

                                        state
 <= SPREAD6;

                                        tsc6Start                       <=
1'd1;



                                end else begin
                                                    //No more spreads can
be formed, add up spread 1, 2, 3, 4 and 5

                                        state
 <= SPREADTOTAL;

                                        tscDone
 = 1'd1;



                                end
```

```
                        end


                SPREAD6: begin
                                                        //STATE 9

                                        state
 <= SPREADTOTAL;                                        //All      the      spreads
calculated, proceed to add them

                                        tscDone
 = 1'd1;

                                end


                SPREADTOTAL:    begin
                                                        //STATE 10

                                                        state
 <= DONE;

                                                        spreadTotalDone <= 1'd1;
                                                    //All spreads added up

                                                        end


                DONE: begin
                                                        //STATE 11

                                state
 <= DONE;                                                   //Final    state
the FSM terminates in

                        end


                default: state                                      <= IDLE;


            endcase


      end
```

```
end



always_ff @(posedge clk) begin


     if (~reset) begin
     //Preparing for calculation, resetting everything


          TSC <= 0;

          magTier1ShortFinal    <= 0;

          magTier2ShortFinal    <= 0;

          magTier3ShortFinal    <= 0;

          tier1ShortFinal       <= 0;

          tier1LongFinal        <= 0;

          tier2ShortFinal       <= 0;

          tier2LongFinal        <= 0;

          tier3ShortFinal       <= 0;

          tier3LongFinal        <= 0;


          for (i = 0; i < 8; i = i + 1) begin

               tier1Short[i]         <= 0;

               tier1Long[i]          <= 0;

               tier2Short[i]         <= 0;

               tier2Long[i]          <= 0;

               tier3Short[i]         <= 0;

               tier3Long[i]          <= 0; end


          for (i = 0; i < 3; i = i + 1) begin
```

```
                short[i]              <= 0;

                long[i]                    <= 0; end


      end else begin
                //Starting the calculation for Inter Month Spread


                for(i = 0; i < 8; i = i + 1) begin
 //Sorting Positions based upon Maturity


                      tier1Short[i]   <=  ((tierMax[0]  >  maturity[i])  &&
(position[i][15] == 1)) ? position[i] : 1'd0;

                      tier1Long[i]    <=  ((tierMax[0]  >  maturity[i])  &&
(position[i][15] == 0)) ? position[i] : 1'd0;

                      tier2Short[i]   <=  ((tierMax[1]  >  maturity[i])  &&
(tierMax[0] <= maturity[i]) && (position[i][15] == 1)) ? position[i] : 1'd0;

                      tier2Long[i]    <=  ((tierMax[1]  >  maturity[i])  &&
(tierMax[0] <= maturity[i]) && (position[i][15] == 0)) ? position[i] : 1'd0;

                      tier3Short[i]   <=  ((tierMax[2]  >  maturity[i])  &&
(tierMax[1] <= maturity[i]) && (position[i][15] == 1)) ? position[i] : 1'd0;

                      tier3Long[i]    <=  ((tierMax[2]  >  maturity[i])  &&
(tierMax[1] <= maturity[i]) && (position[i][15] == 0)) ? position[i] : 1'd0;



                end




                          //Accumulating all the Longs and Shorts in all
Tiers

                tier1ShortFinal <=   tier1Short[0]   +   tier1Short[1]   +
tier1Short[2] + tier1Short[3] + tier1Short[4] + tier1Short[5] + tier1Short[6]
+ tier1Short[7];

                tier2ShortFinal <=   tier2Short[0]   +   tier2Short[1]   +
tier2Short[2] + tier2Short[3] + tier2Short[4] + tier2Short[5] + tier2Short[6]
+ tier2Short[7];
```

```
                tier3ShortFinal <=   tier3Short[0]   +   tier3Short[1]   +
tier3Short[2] + tier3Short[3] + tier3Short[4] + tier3Short[5] + tier3Short[6]
+ tier3Short[7];

                tier1LongFinal  <=   tier1Long[0]    +   tier1Long[1]    +
tier1Long[2] + tier1Long[3] + tier1Long[4] + tier1Long[5] + tier1Long[6] +
tier1Long[7];

                tier2LongFinal  <=   tier2Long[0]    +   tier2Long[1]    +
tier2Long[2] + tier2Long[3] + tier2Long[4] + tier2Long[5] + tier2Long[6] +
tier2Long[7];

                tier3LongFinal  <=   tier3Long[0]    +   tier3Long[1]    +
tier3Long[2] + tier3Long[3] + tier3Long[4] + tier3Long[5] + tier3Long[6] +
tier3Long[7];




                            //Forming Tier Spread Table

            if (positionsAccumulated) begin



                long[0]                         <= tier1LongFinal[6:0];

                long[1]                         <= tier2LongFinal[6:0];

                long[2]                         <= tier3LongFinal[6:0];

                magTier1ShortFinal    =    (~tier1ShortFinal    +    1'd1);
 //Taking the magnitude of Shorts

                magTier2ShortFinal    = (~tier2ShortFinal + 1'd1);

                magTier3ShortFinal    = (~tier3ShortFinal + 1'd1);

                short[0]                    <= magTier1ShortFinal[6:0];

                short[1]                    <= magTier2ShortFinal[6:0];

                short[2]                    <= magTier3ShortFinal[6:0];



            end


                            //Adding up all the spreads for the final Tier
Spread Charge

            if (tscDone) TSC                <= TSC1 + TSC2 + TSC3 + TSC4 +
TSC5 + TSC6 + out1 + out2 + out3;
```

```
      end//if (reset)


 end//always_ff




 always_comb begin


      inputTSC5Long    = (goSpread4) ? TSC4Long : TSC1Long;        //Checking
if Spread charge 4 is skipped and passing appropriate value to instance
"tier13SpreadCharge"



      inputTSC6Short  = ((goSpread5From4)  ||(~goSpread4  &&  goSpread5))  ?
TSC5Short : TSC3Short; //Checking if Spread charge 5 is skipped and passing
appropriate value to instance "tier23SpreadCharge"

 end




//Modules for Tier Spread Calculation

//Tier Spread Charge 1

 tierSpread      tier11SpreadCharge(.clk(clk),           .reset(tsc123Start),
.long(long[0]),       .short(short[0]),       .spreadCharge(spreadCharge[0]),
.outrightChargeTier1(0),     .outrightChargeTier2(0),     .newLong(TSC1Long),
.newShort(TSC1Short), .spread(TSC1), .outright());



//Tier Spread Charge 2

 tierSpread      tier22SpreadCharge(.clk(clk),           .reset(tsc123Start),
.long(long[1]),       .short(short[1]),       .spreadCharge(spreadCharge[1]),
.outrightChargeTier1(0),     .outrightChargeTier2(0),     .newLong(TSC2Long),
.newShort(TSC2Short), .spread(TSC2), .outright());
```

```
//Tier Sprea

 tierSpread      tier33SpreadCharge(.clk(clk),           .reset(tsc123Start),
.long(long[2]),       .short(short[2]),       .spreadCharge(spreadCharge[2]),
.outrightChargeTier1(0),     .outrightChargeTier2(0),     .newLong(TSC3Long),
.newShort(TSC3Short), .spread(TSC3), .outright());



//Tier Spread Charge 4

 tierSpread      tier12SpreadCharge(.clk(clk),             .reset(tsc4Start),
.long(TSC1Long),      .short(TSC2Short),      .spreadCharge(spreadCharge[3]),
.outrightChargeTier1(outright[0]),         .outrightChargeTier2(outright[1]),
.newLong(TSC4Long), .newShort(TSC4Short), .spread(TSC4), .outright(out1));



//Tier Spread Charge 5

 tierSpread      tier13SpreadCharge(.clk(clk),             .reset(tsc5Start),
.long(inputTSC5Long),   .short(TSC3Short),   .spreadCharge(spreadCharge[4]),
.outrightChargeTier1(outright[0]),         .outrightChargeTier2(outright[2]),
.newLong(TSC5Long), .newShort(TSC5Short), .spread(TSC5), .outright(out2));



//Tier Spread Charge 6

 tierSpread      tier23SpreadCharge(.clk(clk),             .reset(tsc6Start),
.long(TSC2Long),   .short(inputTSC6Short),   .spreadCharge(spreadCharge[5]),
.outrightChargeTier1(outright[1]),         .outrightChargeTier2(outright[2]),
.newLong(TSC6Long), .newShort(TSC6Short), .spread(TSC6), .outright(out3));



endmodule
```

tierSpread.sv

For calculating the spread, another module has been implemented. This module has 6 instances in the interMonthSpreadCharge.sv module.

This module is triggered by the FSM inside the interMonthSpread.sv module. It takes in the Long and Short values between which the spreads need to be formed. It forms the spreads and returns the new Long and Short values.

```
module tierSpread (

 input      logic                clk,

 input      logic                reset,

 input      logic [6:0]     long,

 input      logic [6:0]     short,

 input      logic [7:0]     spreadCharge,

 input      logic [7:0]     outrightChargeTier1,

 input      logic [7:0]     outrightChargeTier2,

 output     logic [6:0]     newLong,

 output     logic [6:0]     newShort,

 output     logic [15:0]    spread,

 output     logic [15:0]    outright

);
```

```
/*****************************************************************************
*****************************/
```

```
/************************************************ BODY
 ************************************************/

/************************************************************************************
*****************************/



 always_ff @(posedge clk) begin


      if (~reset)begin
                           //Preparing for calculation, resetting everything


            spread     <= 0;

            outright <= 0;

            newLong    <= 0;

            newShort <= 0;


      end else begin
                                 //Start spread calculation


            if (long <= short) begin
                        //If Short positions are more than Long Positions,


                  if (long != 0) begin
                              //Skip if there are no Long positions, i.e. no
spreads


                        spread     <= spreadCharge * long;
                        //Number of spreads is equal to number of Long
positions

                        newShort <= short - long;
                        //Subtract Long from Short positions to calculate the
remaining Short positions after spreads have been formed

                        newLong    <= 0;
                                    //The remaining Long positions is zero

                        outright <= outrightChargeTier1 - outrightChargeTier2;
 //Calculating outright only when spreads exist
```

```
                end else begin
                                    //When no spreads are formed, pass input
Longs and Shorts directly to output


                     spread     <= 0;

                     outright <= 0;

                     newShort <= short;

                     newLong    <= long;



                end


           end else begin
                                    //If Long positions are more than Short
positions


                if (short != 0) begin
                     //Skip if there are no Short positions



                     spread     <= spreadCharge * short;
                //Number of spreads is equal to number of Short positions

                     newLong    <= long - short;
                     //Subtract Short from Long positions to calculate the
remaining Long positions after spreads have been formed

                     newShort <= 0;
                                    //The remaining Short positions is zero

                     outright <= outrightChargeTier1 - outrightChargeTier2;
 //Calculating outright only when spreads exist



                end else begin
                                    //When no spreads are formed, pass input
Longs and Shorts directly to output


                     spread     <= 0;
```

```
                        outright <= 0;

                        newShort <= short;

                        newLong    <= long;


                    end


            end


      end


 end


endmodule
```

crossCommodity.sv

This module is also triggered by the top module when its data has arrived.

It calculates the credit that needs to be subtracted from the calculated risk margin. It does that by taking the outright rate of the commodity and its correlated commodity along with their ratios. Based on that it calculates the outright margin. That margin is then multiplied with the inter rate which is just a percentage that specifies the effect the calculated outright margin has on the cross commodity charge.

```
module crossComm (

 input      logic                clk,

 input      logic                reset,
```

```
 input      logic [15:0]    outrightRate [0:1],

 input      logic [7:0]     ratio[0:1],

 input      logic [7:0]     interRate,

 output     logic [15:0]    crossCommCharge

);




/*************************************************************************
*****************************/

/*******************************************  Declarations
 ***************************************/

/*************************************************************************
*****************************/



 reg [15:0] outrightMargin;                 //An  intermediate  variable  for
calculation

 reg [31:0] crossCommCharge100;         //Final  Cross  Commodity  charge  in
multiple of 100







/*************************************************************************
*****************************/

/************************************************ BODY
 ************************************************/

/*************************************************************************
*****************************/



 always_ff@(posedge clk) begin


      if(~reset)  begin

 //Preparing for calculation, resetting everything
```

```
            outrightMargin        = 0;

            crossCommCharge100    = 0;

            crossCommCharge       = 0;


      end else begin

 //Starting Cross Commodity Charge calculation


            outrightMargin        =    (outrightRate[0]    *    ratio[0])    +
(outrightRate[1] * ratio[1]);          //The  Outright  Marging  as  per  the
ratios between 2 commodities

            crossCommCharge100    = interRate * outrightMargin;
                                                  //Inter  rate  selects  the
percentage of Outright Margin that will influence CrossCommCharge

            crossCommCharge       = (crossCommCharge100 / 100 );
                                            //Dividing  by  100  to  get  the
final Cross Commodity Charge



      end


  end


 endmodule
```

Test Benches

These are the testbeches used for testing the code in ModelSim.

test.sv

```
module test_bench;

 logic clk;

 logic reset;

 logic [15:0] writeData;

 logic [5:0] offset;

 logic  write;

 logic  chipselect;


 logic [15:0] priceScanRange;

 logic [15:0] readData;

 logic read;
```

```
 integer i;


span_cme span_cme1(.*);


initial begin

clk = 0;

 forever # 1 clk = ~ clk;

end









initial begin

clk = 0;

reset = 1;

write = 0;

chipselect = 0;

@(posedge clk) ;

reset =0 ;

write = 1;

chipselect = 1;

@(posedge clk);

writeData = 16'd96;    //Price Scan Range

offset = 6'd0;

@(posedge clk);

offset = 6'd1;

writeData = 32'd25;    //Position 1

@(posedge clk);
```

```
offset = 6'd2;

writeData = -32'd5;    //Position 2

@(posedge clk);

offset = 6'd3;

writeData = 32'd10;    //Position 3

@(posedge clk);

offset = 6'd4;

writeData = -32'd15;   //Position 4

@(posedge clk);

offset = 6'd5;

writeData = 32'd5;     //Position 5

@(posedge clk);

offset = 6'd6;

writeData = -32'd15;   //Position 6

@(posedge clk);

offset = 6'd7;

writeData = 32'd0;     //Position 7

@(posedge clk);

offset = 6'd8;

writeData = 32'd0;     //Position 8

@(posedge clk);

offset = 6'd9;

writeData= 32'd175;    //Outright rate for 1st commodity

@(posedge clk);

offset = 6'd10;

writeData= 32'd250;    //Outright rate for 2nd commodity

@(posedge clk);

offset = 6'd11;
```

```
writeData= 32'd2;      //Cross commodity ratio for 1st commodity

@(posedge clk);

offset = 6'd12;

writeData= 32'd1;      //Cross commodity ratio for 2nd commodity

@(posedge clk);

offset = 6'd13;

writeData= 32'd55;     //Cross commodity correlation factor

@(posedge clk);

offset = 6'd14;

writeData = 32'd1;     //Maturity for Position 1

@(posedge clk);

offset = 6'd15;

writeData = 32'd1;     //Maturity for Position 2

@(posedge clk);

offset = 6'd16;

writeData = 32'd3;     //Maturity for Position 3

@(posedge clk);

offset = 6'd17;

writeData = 32'd3;     //Maturity for Position 4

@(posedge clk);

offset = 6'd18;

writeData = 32'd5;     //Maturity for Position 5

@(posedge clk);

offset = 6'd19;

writeData = 32'd5;     //Maturity for Position 6

@(posedge clk);

offset = 6'd20;

writeData = 32'd0;     //Maturity for Position 7
```

```
@(posedge clk);
offset = 6'd21;
writeData = 32'd0;     //Maturity for Position 8
@(posedge clk);
offset = 6'd22;
writeData = 32'd2;     //Tier 1 upper limit
@(posedge clk);
offset = 6'd23;
writeData = 32'd4;     //Tier 2 upper limit
@(posedge clk);
offset = 6'd24;
writeData = 32'd6;     //Tier 3 upper limit
@(posedge clk);
offset = 6'd25;
writeData = 32'd50;    //Spread Charge 1
@(posedge clk);
offset = 6'd26;
writeData = 32'd60;    //Spread Charge 2
@(posedge clk);
offset = 6'd27;
writeData = 32'd70;    //Spread Charge 3
@(posedge clk);
offset = 6'd28;
writeData = 32'd80;    //Spread Charge 4
@(posedge clk);
offset = 6'd29;
writeData = 32'd90;    //Spread Charge 5
@(posedge clk);
```

```
offset = 6'd30;

writeData = 32'd100;   //Spread Charge 6

@(posedge clk);

offset = 6'd31;

writeData = 32'd100;   //Outrigt rate 1

@(posedge clk);

offset = 6'd32;

writeData = 32'd110;   //Outrigt rate 2

@(posedge clk);

offset = 6'd33;

writeData= 32'd120;    //Outrigt rate 3



for (i=0 ; i<200; i=i+1)

    begin

            @(posedge clk);

    end


end


endmodule
```

test2.sv

```
module test_bench;
 logic clk;
 logic reset;
 logic [15:0] writeData;
 logic [5:0] offset;
 logic  write;
 logic  chipselect;

 logic [15:0] priceScanRange;
 logic [15:0] readData;
 logic read;
 integer i;

span_cme span_cme1(.*);

initial begin
clk = 0;
 forever # 1 clk = ~ clk;
end
```

```
initial begin

clk = 0;

reset = 1;

write = 0;

chipselect = 0;

@(posedge clk) ;

reset =0 ;

write = 1;

chipselect = 1;

@(posedge clk);

writeData = 16'd100;   //Price Scan Range

offset = 6'd0;

@(posedge clk);

offset = 6'd1;

writeData = 32'd25;    //Position 1

@(posedge clk);

offset = 6'd2;

writeData = -32'd5;    //Position 2

@(posedge clk);

offset = 6'd3;

writeData = 32'd15;    //Position 3

@(posedge clk);

offset = 6'd4;

writeData = -32'd15;   //Position 4

@(posedge clk);

offset = 6'd5;

writeData = 32'd5;     //Position 5

@(posedge clk);
```

```
offset = 6'd6;
writeData = -32'd20;   //Position 6
@(posedge clk);
offset = 6'd7;
writeData = 32'd15;    //Position 7
@(posedge clk);
offset = 6'd8;
writeData = 32'd5;     //Position 8
@(posedge clk);
offset = 6'd9;
writeData= 32'd100;    //Outright rate for 1st commodity
@(posedge clk);
offset = 6'd10;
writeData= 32'd200;    //Outright rate for 2nd commodity
@(posedge clk);
offset = 6'd11;
writeData= 32'd2;      //Cross commodity ratio for 1st commodity
@(posedge clk);
offset = 6'd12;
writeData= 32'd1;      //Cross commodity ratio for 2nd commodity
@(posedge clk);
offset = 6'd13;
writeData= 32'd50;     //Cross commodity correlation factor
@(posedge clk);
offset = 6'd14;
writeData = 32'd1;     //Maturity for Position 1
@(posedge clk);
offset = 6'd15;
```

```
writeData = 32'd1;     //Maturity for Position 2

@(posedge clk);

offset = 6'd16;

writeData = 32'd3;     //Maturity for Position 3

@(posedge clk);

offset = 6'd17;

writeData = 32'd3;     //Maturity for Position 4

@(posedge clk);

offset = 6'd18;

writeData = 32'd5;     //Maturity for Position 5

@(posedge clk);

offset = 6'd19;

writeData = 32'd5;     //Maturity for Position 6

@(posedge clk);

offset = 6'd20;

writeData = 32'd3;     //Maturity for Position 7

@(posedge clk);

offset = 6'd21;

writeData = 32'd5;     //Maturity for Position 8

@(posedge clk);

offset = 6'd22;

writeData = 32'd2;     //Tier 1 upper limit

@(posedge clk);

offset = 6'd23;

writeData = 32'd4;     //Tier 2 upper limit

@(posedge clk);

offset = 6'd24;

writeData = 32'd6;     //Tier 3 upper limit
```

```
@(posedge clk);

offset = 6'd25;

writeData = 32'd50;    //Spread Charge 1

@(posedge clk);

offset = 6'd26;

writeData = 32'd60;    //Spread Charge 2

@(posedge clk);

offset = 6'd27;

writeData = 32'd70;    //Spread Charge 3

@(posedge clk);

offset = 6'd28;

writeData = 32'd80;    //Spread Charge 4

@(posedge clk);

offset = 6'd29;

writeData = 32'd90;    //Spread Charge 5

@(posedge clk);

offset = 6'd30;

writeData = 32'd100;   //Spread Charge 6

@(posedge clk);

offset = 6'd31;

writeData = 32'd100;   //Outrigt rate 1

@(posedge clk);

offset = 6'd32;

writeData = 32'd110;   //Outrigt rate 2

@(posedge clk);

offset = 6'd33;

writeData= 32'd120;    //Outrigt rate 3
```

```
for (i=0 ; i<200; i=i+1)
    begin
            @(posedge clk);
    end

end

endmodule
```

test3.sv

```
module test_bench;
 logic clk;
 logic reset;
 logic [15:0] writeData;
 logic [5:0] offset;
```

```
 logic  write;
 logic  chipselect;

 logic [15:0] priceScanRange;
 logic [15:0] readData;
 logic read;
 integer i;

span_cme span_cme1(.*);

initial begin
clk = 0;
 forever # 1 clk = ~ clk;
end

initial begin
clk = 0;
reset = 1;
write = 0;
chipselect = 0;
@(posedge clk) ;
reset =0 ;
write = 1;
chipselect = 1;
@(posedge clk);
```

```
writeData = 16'd200;   //Price Scan Range

offset = 6'd0;

@(posedge clk);

offset = 6'd1;

writeData = 32'd25;    //Position 1

@(posedge clk);

offset = 6'd2;

writeData = -32'd5;    //Position 2

@(posedge clk);

offset = 6'd3;

writeData = 32'd10;    //Position 3

@(posedge clk);

offset = 6'd4;

writeData = -32'd15;   //Position 4

@(posedge clk);

offset = 6'd5;

writeData = 32'd5;     //Position 5

@(posedge clk);

offset = 6'd6;

writeData = -32'd15;   //Position 6

@(posedge clk);

offset = 6'd7;

writeData = 32'd0;     //Position 7

@(posedge clk);

offset = 6'd8;

writeData = 32'd0;     //Position 8

@(posedge clk);

offset = 6'd9;
```

```
writeData= 32'd175;    //Outright rate for 1st commodity

@(posedge clk);

offset = 6'd10;

writeData= 32'd250;    //Outright rate for 2nd commodity

@(posedge clk);

offset = 6'd11;

writeData= 32'd2;      //Cross commodity ratio for 1st commodity

@(posedge clk);

offset = 6'd12;

writeData= 32'd1;      //Cross commodity ratio for 2nd commodity

@(posedge clk);

offset = 6'd13;

writeData= 32'd55;     //Cross commodity correlation factor

@(posedge clk);

offset = 6'd14;

writeData = 32'd1;     //Maturity for Position 1

@(posedge clk);

offset = 6'd15;

writeData = 32'd1;     //Maturity for Position 2

@(posedge clk);

offset = 6'd16;

writeData = 32'd3;     //Maturity for Position 3

@(posedge clk);

offset = 6'd17;

writeData = 32'd3;     //Maturity for Position 4

@(posedge clk);

offset = 6'd18;

writeData = 32'd5;     //Maturity for Position 5
```

```
@(posedge clk);
offset = 6'd19;
writeData = 32'd5;     //Maturity for Position 6
@(posedge clk);
offset = 6'd20;
writeData = 32'd0;     //Maturity for Position 7
@(posedge clk);
offset = 6'd21;
writeData = 32'd0;     //Maturity for Position 8
@(posedge clk);
offset = 6'd22;
writeData = 32'd2;     //Tier 1 upper limit
@(posedge clk);
offset = 6'd23;
writeData = 32'd4;     //Tier 2 upper limit
@(posedge clk);
offset = 6'd24;
writeData = 32'd6;     //Tier 3 upper limit
@(posedge clk);
offset = 6'd25;
writeData = 32'd50;    //Spread Charge 1
@(posedge clk);
offset = 6'd26;
writeData = 32'd60;    //Spread Charge 2
@(posedge clk);
offset = 6'd27;
writeData = 32'd70;    //Spread Charge 3
@(posedge clk);
```

```
offset = 6'd28;
writeData = 32'd80;    //Spread Charge 4
@(posedge clk);
offset = 6'd29;
writeData = 32'd90;    //Spread Charge 5
@(posedge clk);
offset = 6'd30;
writeData = 32'd100;   //Spread Charge 6
@(posedge clk);
offset = 6'd31;
writeData = 32'd100;   //Outrigt rate 1
@(posedge clk);
offset = 6'd32;
writeData = 32'd110;   //Outrigt rate 2
@(posedge clk);
offset = 6'd33;
writeData= 32'd120;    //Outrigt rate 3



for (i=0 ; i<200; i=i+1)
    begin
            @(posedge clk);
    end

end

endmodule
```

Sample TCL script

test.tcl(Its values are the same as test.sv testbench file)

```
# A Tcl script for the Qsys system console


# Start Qsys, open your soc_system.qsys file, run File->System Console,
# then execute this script by selecting it with Ctrl-E


# The System Console is described in Chapter 10 of Volume III of
# the Quartus II Handbook


# Alternately,
# system-console --project_dir=. --script=syscon-test.tcl
#
# system-console --project_dir=. -cli
#   and then "source syscon-test.tcl"


# Base addresses of the peripherals: take from Qsys
set vga_led 0x0


puts "Started system-console-test-script"
```

```
# Using the JTAG chain, check the clock and reset"

set j [lindex [get_service_paths jtag_debug] 0]
open_service jtag_debug $j
puts "Opened jtag_debug"

puts "Checking the JTAG chain loopback: [jtag_debug_loop $j {1 2 3 4 5 6}]"
jtag_debug_reset_system $j

puts -nonewline "Sampling the clock: "
foreach i {1 1 1 1 1 1 1 1 1 1 1} {
   puts -nonewline [jtag_debug_sample_clock $j]
}
puts ""

puts "Checking reset state: [jtag_debug_sample_reset $j]"

close_service jtag_debug $j
puts "Closed jtag_debug"

# Perform bus reads and writes

set m [lindex [get_service_paths master] 0]
open_service master $m
puts "Opened master"
```

Write a test pattern to the various registers

#foreach { r v } {0 0xff 1 0x1 2 0x2 3 0x4 4 0x8 5 0x10 6 0x20 7 0x40} {

foreach { r v } {

0 96

2 25

4 -5

6 10

8 -15

10 5

12 -15

14 0

16 0

18 175

20 250

22 2

24 1

26 55

28 1

30 1

32 3

34 3

36 5

38 5

40 0

42 0

44 2

46 4

```
48 6

50 50

52 60

54 70

56 80

58 90

60 100

62 100

64 110

66 120} {

    master_write_16 $m [expr $vga_led + $r] $v

    puts $v

}

puts "read finished -sleep start"

#after 5000

puts "woken up - read start"

foreach { i } { 0 2 4 6 8} {

        puts $i

        puts [ master_read_16 $m [expr $vga_led + $i ] 1]


}


close_service master $m

puts "Closed master"
```

C) Span_cme.c

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "span_cme.h"

#define DRIVER_NAME "span_cme"

/*
 * Information about our device
 */
struct span_cme_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;

/*
 * Writes the input data  of a single portifolio to the peripheral
 * works serially, one data entry at a time
 */
static void write_digit(short input[])//, short input1)
{
    int it;
    for ( it = 0; it < DATA_LENGTH; it++) {
      iowrite16(input[it], dev.virtbase + (2*it));
    }
}
/* read the out put data from the board,
 * can be modified to include different reads for the debug data (individual
parameter values) as well
 */
static short read_digit()
{
    return  ioread16(dev.virtbase);
}
/*
 * Handle ioctl() calls from userspace:
 * Read or write  data from the portifolio.
 *
 */
static long span_cme_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    portifolio_t vla;
```

```
    switch (cmd) {
    case SPAN_CME_WRITE_DIGIT:
      if (copy_from_user(&vla, (portifolio_t *) arg,
                         sizeof(portifolio_t)))
            return -EACCES;
      write_digit(vla.input);
      break;

    case SPAN_CME_READ_DIGIT:
      if (copy_from_user(&vla, (portifolio_t *) arg,
                         sizeof(portifolio_t)))
            return -EACCES;
      vla.output = read_digit();
      if (copy_to_user((portifolio_t *) arg, &vla,
                       sizeof(portifolio_t)))
            return -EACCES;
      break;

    default:
      return -EINVAL;
    }
    return 0;
}

/* The operations our device knows how to do */
static const struct file_operations span_cme_fops = {
    .owner      = THIS_MODULE,
    .unlocked_ioctl = span_cme_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice span_cme_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &span_cme_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init span_cme_probe(struct platform_device *pdev)
{
    int ret;

    /* Register ourselves as a misc device: creates /dev/span_cme */
    ret = misc_register(&span_cme_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
      ret = -ENOENT;
      goto out_deregister;
    }

    /* Make sure we can use these registers */
```

```
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
                           DRIVER_NAME) == NULL) {
      ret = -EBUSY;
      goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
      ret = -ENOMEM;
      goto out_release_mem_region;
    }
    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&span_cme_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int span_cme_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&span_cme_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id span_cme_of_match[] = {
    { .compatible = "altr,span_cme" },
    {},
};
MODULE_DEVICE_TABLE(of, span_cme_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver span_cme_driver = {
    .driver     = {
      .name = DRIVER_NAME,
      .owner     = THIS_MODULE,
      .of_match_table = of_match_ptr(span_cme_of_match),
    },
    .remove     = __exit_p(span_cme_remove),
};

/* Called when the module is loaded: set things up */
static int __init span_cme_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&span_cme_driver, span_cme_probe);
}
```

```c
/* Called when the module is unloaded: release resources */
static void __exit span_cme_exit(void)
{
    platform_driver_unregister(&span_cme_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(span_cme_init);
module_exit(span_cme_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Vidhatre Gathey, CME, CSEE4840, Columbia University");
MODULE_DESCRIPTION("kernel interface with span_cme module");
```

D)

```c
/*
 * -Userspace program that communicates with the span_cme peripheral
 *  primarily through ioctls
 * -program reads the portifolio data from the "input_file.txt"
 * -program outputs all the read data from the peripheral to "output_file.txt"
 *
 * Vidhatre Gathey
 * Columbia University
 */

#include <stdio.h>
#include "span_cme.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <inttypes.h>
#define NUM_OF_PORTIFOLIO 3 // number of portifolios the input contains
int span_cme_fd;

/* function to read the output from the board using an ioctl function,
 * open an output file, and
 * append the output data to the file
 */
void print_output() {
    portifolio_t port;
    FILE *fpout;
    fpout = fopen("output_file.txt","a");

    if (ioctl(span_cme_fd, SPAN_CME_READ_DIGIT, &port)) {
      perror("ioctl(SPAN_CME_READ_DIGIT) failed");
      return;
    }
     printf("%04d ", port.output);
    fprintf(fpout,"%04d ", port.output);
```

```
    printf("\n");
    fprintf(fpout,"\n");

    if (fpout) fclose(fpout);
}

/* Write the contents of the input portifolio data to the board,
 * used the ioctl function to wrtie to the board
 */
void write_portifolio(short var[])
{
  portifolio_t port;
  int i;
    for (i=0; i< DATA_LENGTH; i++)
     port.input[i] = var[i];
    port.output = 0;
    if (ioctl(span_cme_fd, SPAN_CME_WRITE_DIGIT, &port)) {
      perror("ioctl(SPAN_CME_WRITE_DIGIT) failed");
      return;
    }
}

int main()
{
  portifolio_t port;       // portifolio template as described in the header
span_cme.h
  int i,j,k;               // iterators
  short var[DATA_LENGTH];      // used as a buffer for storing the data from the
input file as it is read
  char label[DATA_LENGTH][14];  // store the first line of labels from the input
file
  static const char filename[] = "/dev/span_cme";
  FILE *fpin;

  printf("SPAN CME Userspace program started\n");

  fpin=fopen("input_file.txt", "r+"); // open input file

  if ( (span_cme_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }

  for (j = NUM_OF_PORTIFOLIO ; j > 0 ; j--){   // read the a set of data i.e. a
portifolio from the input file
    for(k = 0; k < DATA_LENGTH; k++)
      fscanf( fpin, "%hu",&var[k]);
    write_portifolio(var);
    print_output();
  }
  fclose(fpin);

  printf("SPAN CME Userspace program terminating\n");
  return 0;
}
```

E) SPAN_CME.h

```
#ifndef _SPAN_CME_H
#define _SPAN_CME_H

#include <linux/ioctl.h>
#define DATA_LENGTH 34 // data length for a portifolio is defined here

typedef struct {      // currently the portifolio struct just hold the input data
and the calculated risk margin
    short input[DATA_LENGTH];
    short output;
} portifolio_t; //portifolio_t;


#define SPAN_CME_MAGIC 'q'

/* ioctls and their arguments */
#define SPAN_CME_WRITE_DIGIT _IOW(SPAN_CME_MAGIC, 1, portifolio_t *)
#define SPAN_CME_READ_DIGIT  _IOWR(SPAN_CME_MAGIC, 2, portifolio_t *)

#endif
```