Cox-Ross-Rubinstein (CRR) Binomial Tree Pricing Model

by Mourad Musallam

ELECTRICAL [+] COMPUTER

ENGINEERING

Georgia Tech School of Electrical and Computer Engineering

CRR Binomial Option

- Parameter inputs
  - Number of Steps: n = 50
  - Spot Price: Spot = 100
  - Strike Price: K = 100
  - Expiration Date: T = 1 year
  - Dividend Yield: q = 0.05
  - Volatility: v =0.20
  - Risk Free Rate: r = 0.05

  - $\Delta t = T/n$
  - Up factor: u = $e^{v*\sqrt{\Delta t}}$
  - Down factor: d = 1/u
  - Probability: p = $\frac{e^{(r-q)(\Delta t)} - d}{u-d}$

Output: The price of American and European option at a specific instant in time

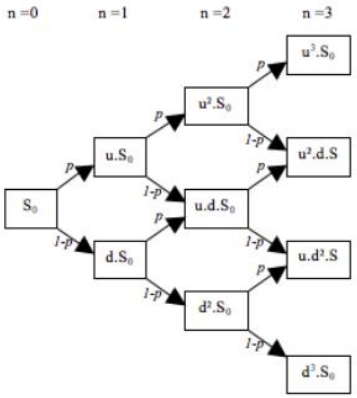
AC: American Call Option

AP: American Put Option

EC: European Call Option

EP: European Put Option

Step 1:Build Price Tree



Step 2: Option value at each final node

$Max[(S_n - K), 0]$ *Call option*

$Max[K - S_n), 0]$ *Put option*

Step 3: Option value at previous nodes

Binomial Value= $[p \times option\ up + (1-p) \times option\ down] \times e^{-r\Delta t}$

Main

- Pass in your price tree array and four option pricing arrays into top function.
  - AC: American Call Option
  - AP: American Put Option
  - EC: European Call Option
  - EP: European Put Option

```
int main()
{

    float S[n+1][n+1];
    float EC[n+1][n+1];
    float EP[n+1][n+1];
    float AC[n+1][n+1];
    float AP[n+1][n+1];

    top(AC,AP,EC,EP,S);

// Output of prices of calls and puts

    cout << "The Cox Ross Rubinstein prices using " << n << " steps are... " << endl;

    cout << "European Call " << EC[0][0] << endl;

    cout << "European Put  " << EP[0][0] << endl;

    cout << "American Call " << AC[0][0] << endl;

    cout << "American Put  " << AP[0][0] << endl;

    cout << endl;
    return 0;

}
```

Build Price Tree Optimization

- This function builds the binomial tree using the up/down factor.
- Loop re-ordering and indexing. Original loop order caused a huge increase in latency.
- This allowed for the price tree to properly unroll.

```
void buildtree(float mat[n+1][n+1])
{

    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            #pragma HLS unroll
            mat[j][i] = Spot*powf(u, i - j)*powf(d, j);
        }
    }
}
```

Terminal Pay Off Node Optimization

- This function computes final payoff at the final nodes of tree.
- Could not optimize except for in lining the max function.
- Potential improvements
  - Don’t use matrix for binomial tree.
  - Use a struct to build tree could allow improvement.

```
void terminalPayoff(float matAC[n+1][n+1],float matAP[n+1][n+1],float matEC[n+1][n+1],
float matEP[n+1][n+1], float matS[n+1][n+1])
{

    for (int i = 0; i <= n; i++)
    {
        matEC[i][n] = max(matS[i][n] - K, 0.0);
        matAC[i][n] = max(matS[i][n] - K, 0.0);
        matEP[i][n] = max(K - matS[i][n], 0.0);
        matAP[i][n] = max(K - matS[i][n], 0.0);
    }
}
```

Backward Recursion Optimization

- This function computes option value at all earlier nodes of the tree.
- Nested loops did not allow for any improvement.
- Make the inner loop into a function.
- Apply pragma dataflow to the function. Led to pipeline of the inner loop.
- Report showed the inner loop is pipelined but I am doubtful on this.

```
void backwardRecursion(float matAC[n+1][n+1],float matAP[n+1][n+1],float matEC[n+1][n+1],
float matEP[n+1][n+1], float matS[n+1][n+1])
{
    for (int j = n - 1; j >= 0; j--)
    {
        #pragma HLS dataflow
        subfunction(matAC,matAP,matEC,matEP,matS,j);
    }
}
```

```
void subfunction(float matAC[n+1][n+1],float matAP[n+1][n+1],
float matEC[n+1][n+1],float matEP[n+1][n+1], float matS[n+1][n+1],int k)
{
    for(int i = 0;i<=k;i++)
    {
        matEC[i][k] =expf(-r*dt)*(p*matEC[i][k + 1] + (1 - p)*matEC[i + 1][k + 1]);
        matEP[i][k] =expf(-r*dt)*(p*matEP[i][k + 1] + (1 - p)*matEP[i + 1][k + 1]);
        matAC[i][k] = max(matS[i][k] - K,expf(-r*dt)*(p*matAC[i][k + 1] + (1 - p)*matAC[i + 1][k + 1]));
        matAP[i][k] = max(K - matS[i][k],expf(-r*dt)*(p*matAP[i][k + 1] + (1 - p)*matAP[i + 1][k + 1]));
    }
}
```

Summary of Optimizations

```
void buildtree(float mat[n+1][n+1])
{

    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            #pragma HLS unroll
            mat[j][i] = Spot*powf(u, i - j)*powf(d, j);
        }
    }
}
```

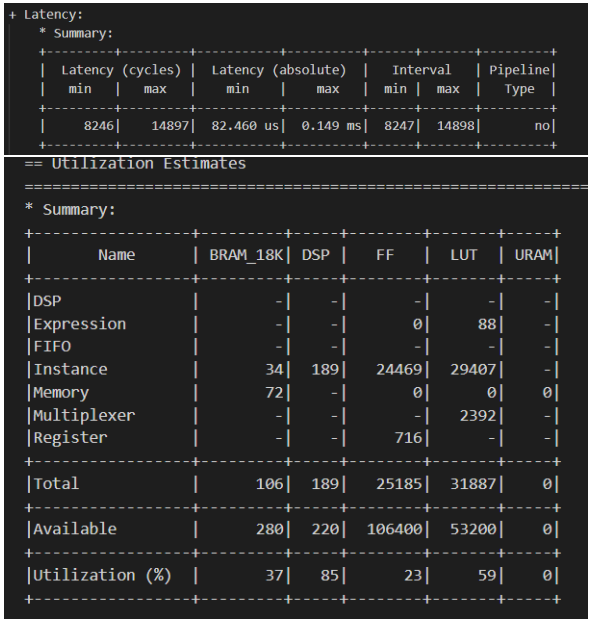
```
void terminalPayoff(float matAC[n+1][n+1],float matAP[n+1][n+1],float matEC[n+1][n+1],
float matEP[n+1][n+1], float matS[n+1][n+1])
{

    for (int i = 0; i <= n; i++)
    {
        matEC[i][n] = max(matS[i][n] - K, 0.0);
        matAC[i][n] = max(matS[i][n] - K, 0.0);
        matEP[i][n] = max(K - matS[i][n], 0.0);
        matAP[i][n] = max(K - matS[i][n], 0.0);
    }
}
```

```
void backwardRecursion(float matAC[n+1][n+1],float matAP[n+1][n+1],float matEC[n+1][n+1],
float matEP[n+1][n+1], float matS[n+1][n+1])
{
    for (int j = n - 1; j >= 0; j--)
    {
        #pragma HLS dataflow
        subfunction(matAC,matAP,matEC,matEP,matS,j);
    }
}
```

```
void subfunction(float matAC[n+1][n+1],float matAP[n+1][n+1],
float matEC[n+1][n+1],float matEP[n+1][n+1], float matS[n+1][n+1],int k)
{
    for(int i = 0;i<=k;i++)
    {
        matEC[i][k] =expf(-r*dt)*(p*matEC[i][k + 1] + (1 - p)*matEC[i + 1][k + 1]);
        matEP[i][k] =expf(-r*dt)*(p*matEP[i][k + 1] + (1 - p)*matEP[i + 1][k + 1]);
        matAC[i][k] = max(matS[i][k] - K,expf(-r*dt)*(p*matAC[i][k + 1] + (1 - p)*matAC[i + 1][k + 1]));
        matAP[i][k] = max(K - matS[i][k],expf(-r*dt)*(p*matAP[i][k + 1] + (1 - p)*matAP[i + 1][k + 1]));
    }
}
```

- Loop Ordering
- Array Partition
- Unrolling
- Dataflow
- Pipelining
- Inline

Results



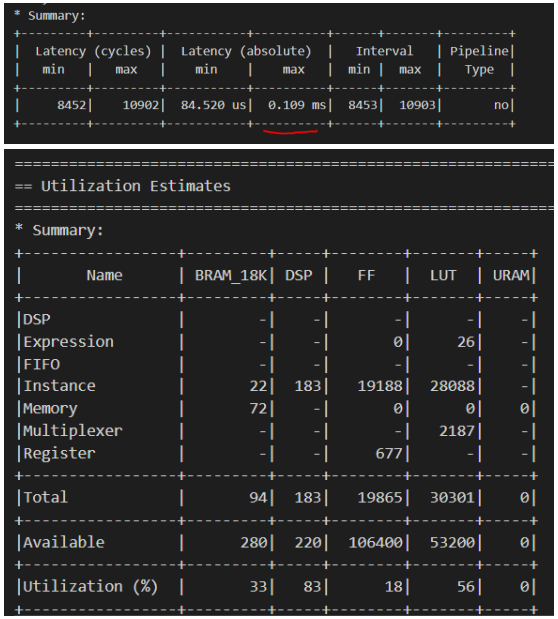
+ Latency:
    * Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
8246	14897	82.460 us	0.149 ms	8247	14898	no

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	88	-
FIFO	-	-	-	-	-
Instance	34	189	24469	29407	-
Memory	72	-	0	0	0
Multiplexer	-	-	-	2392	-
Register	-	-	716	-	-
Total	106	189	25185	31887	0
Available	280	220	106400	53200	0
Utilization (%)	37	85	23	59	0



* Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
8452	10902	84.520 us	0.109 ms	8453	10903	no

== Utilization Estimates

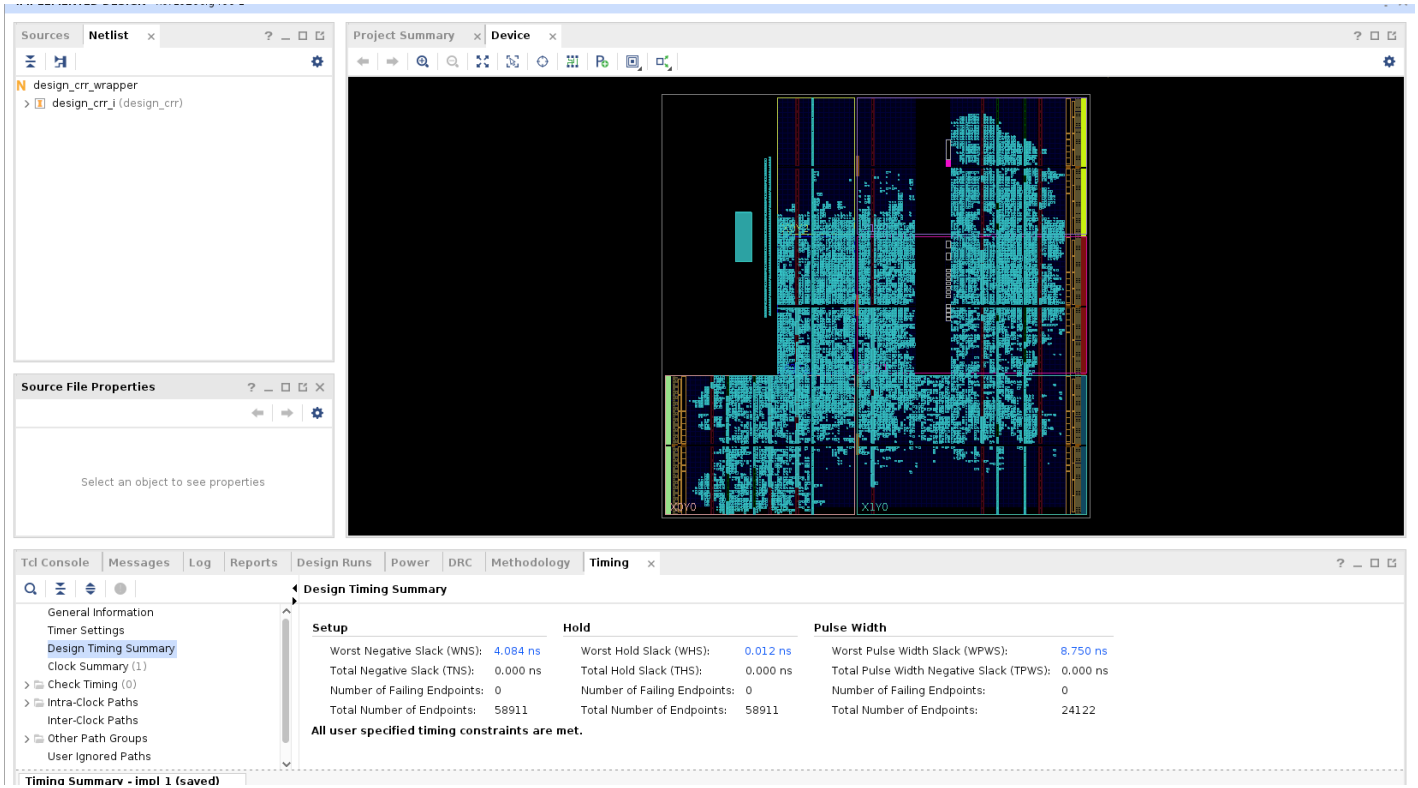
* Summary:

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	26	-
FIFO	-	-	-	-	-
Instance	22	183	19188	28088	-
Memory	72	-	0	0	0
Multiplexer	-	-	-	2187	-
Register	-	-	677	-	-
Total	94	183	19865	30301	0
Available	280	220	106400	53200	0
Utilization (%)	33	83	18	56	0

- Note: This is for 5 memory ports on a zync board the zync board were using has only 4 this will most likely change.
- Speed-up = 1.36X
  - Output

    The Cox Ross Rubinstein prices using 50 steps are...
    European Call 7.53931
    European Put 7.53927
    American Call 7.63081
    American Put 7.63077

Vivado output



Future works or potential improvement

- Complete code restructure, try a different code structure ie structs vs arrays (may try this before deadline). This may lead to improvements.
- Try different tree structure styles (this makes a difference).
- Use fixed point vs float
- If the current setup is ok then this can be used across different stocks.
- Further optimize current code and scale for large “n” values.
- Compare with Black Scholes Model.

Thank you