Computational Model for Re-entrant Multiple Hardware Threads

Rakhee Keswani

University of Kansas

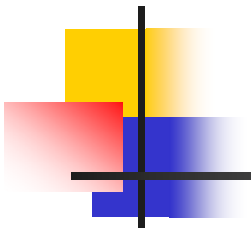
21st July 2005

Committee Members

Dr. Daniel Deavours

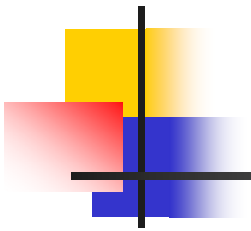
Dr. Perry Alexander

Dr. James Stiles



Outline

- **Introduction**
- **Computational Model**
- **Examples of Transformations**
- **Fibonacci Example**
- **Results and Future Work**



Introduction

- One of the biggest problems in computing is to process large quantities of data in the minimum time with minimum levels of power consumption
- Reconfigurable computing offers better price/performance ratio over COTS components
- Programming hybrid devices in such a way as to maximize the usage of available resources is difficult
- The threaded programming model is established as a mechanism for handling the interactions of concurrent, lightweight computations
- The proposed research addresses the question of how to efficiently map a threaded programming model onto a computational model for modern FPGAs.

Related Work

- HARDWAREC is a fully synthesizable language with a C-like syntax .It doesn't support pointers, recursion and dynamic memory allocation.
- CONES is an automated synthesis system that takes behavioral models written in a C-based language and produces gate-level implementations. This subset is very restricted and doesn't contain unbounded loops nor pointers.
- SYSTEMC supports a mixed synchronous and asynchronous approach implemented as a C++ library.
- Other extensions include ECL from Cadence based on C and Esterel, HANDEL-C

Computational Model
Types of Transformations

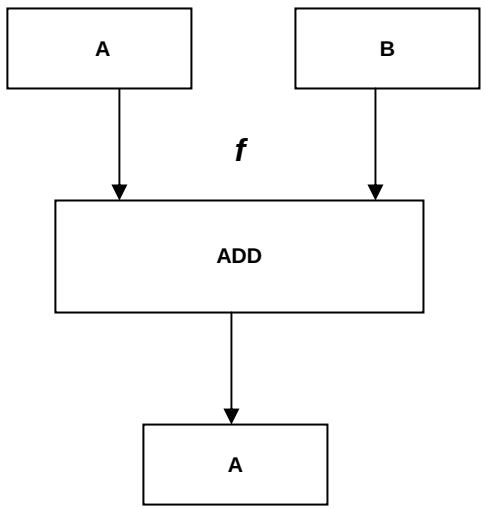
A computational model is defined informally as a systematic, coherent framework for computation.

A transformation is an atomic unit of computational model roughly analogous to a machine instruction or a set of instructions.

- Simple transformations
- Routing transformations
- Dual transformations
- First-in-First-out (FIFO) transformation

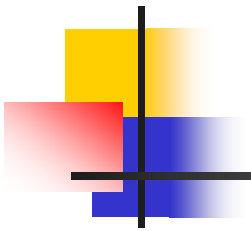
➢ Flow control and Scheduling

Simple Transformations



A=add(A,B)

- f: A→B
- The set of functions f could be, is constrained by the capabilities of hardware and timing requirements
- Simple transformations make up the primary data computations in the system

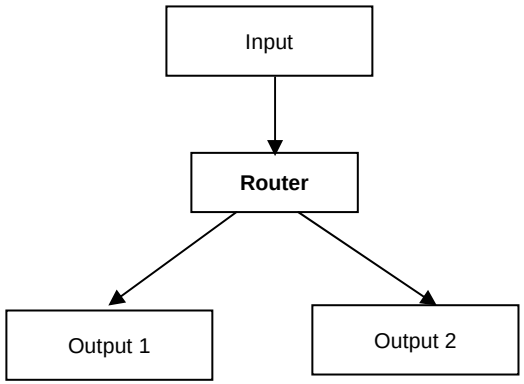


Routing Transformations

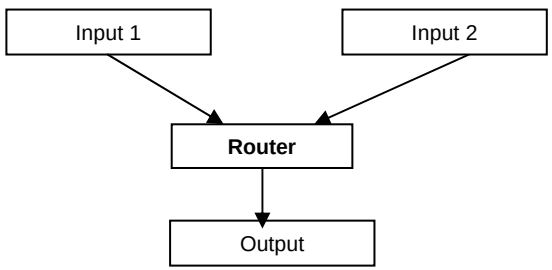
- Routers are structures that route thread states between other transformations. They are analogous to, but different from, branch instructions in traditional processes.

Contd..

Routing Transformations

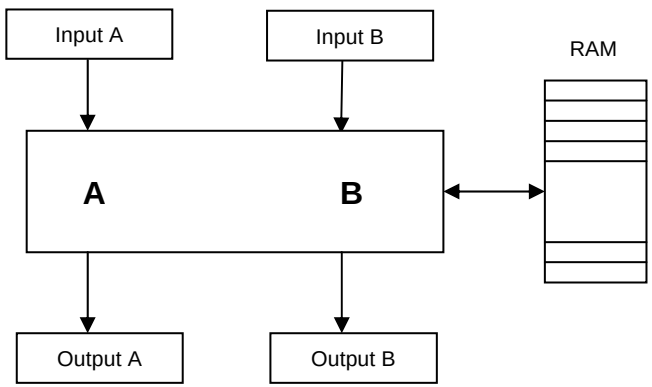



```
process(input,selector)
begin
  if (selector ='0') then
    output 1 <= input;
  else output 2 <=input;
  end if;
end process;
```




```
process(input1,input2,selector)
begin
      if (selector ='0') then
           output<= input 1;
      else output<= input 2;
      end if;
end process;
```

Dual Transformations



General form of Dual Transformation

Example

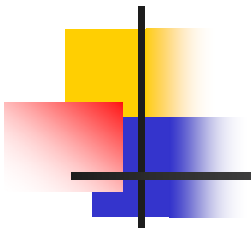
CALL

- Thread at the input of port A
- RAM write (save state)
- New thread containing return information at the output of port A

RETURN

- The function returns at the input of port B
- RAM read
- The saved state of the thread is appended with the function return value and is emitted at the output of port B

Contd..



Dual Transformations

- The mechanism for storing the thread state in the RAM depends on the nature of the function.
- If the function requires that the threads be returned in the same order as they were called, then the RAM can be organized as a simple FIFO.
- If the threads are returned in a random order, then a unique address must be passed along with the thread state.

Contd..

Dual Transformations

- Special logic is required to keep track of addresses of RAM entries that are empty and those that are full.
- One way to do is to assign a flag bit for each address. The flag bit is set to ‘1’ if the address is free and set to ‘0’ if the address is full.
  - Extra memory space to store the flag bits. As the length of the RAM increases the number of flag bits increases.
  - Search algorithm to find out which flag bit is ‘1’ and which is ‘0’.

Contd..

Dual Transformations

- Another implementation is to use a linked list to manage the free memory address.
- Freed address is inserted into the head of the list, and requires one memory write to that address to update the link to the head pointer.

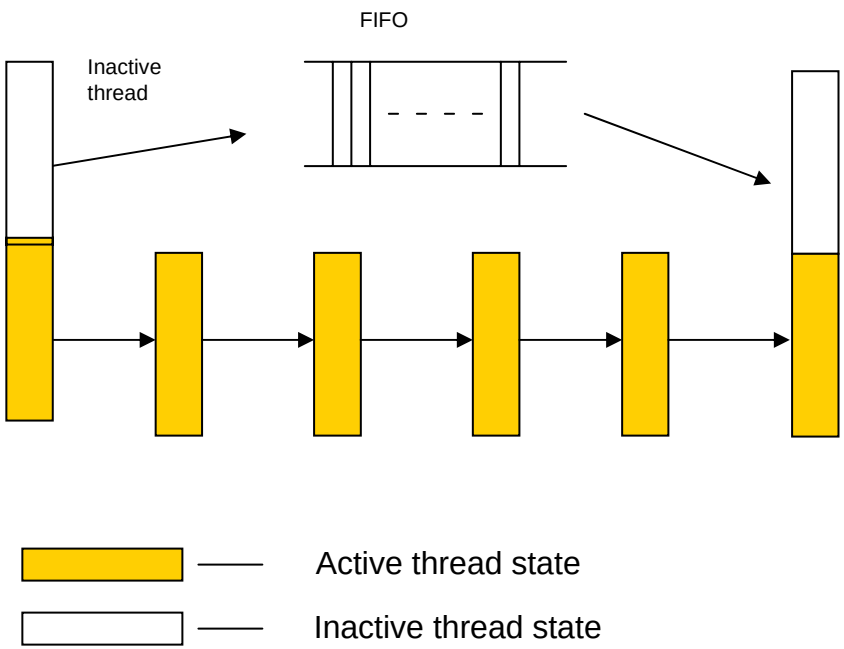
```
free(addr)
    RAM[addr]<= head
    head<=addr
```

- When requested, a free address is allocated from the head of the list, which requires a memory read to update the head pointer.

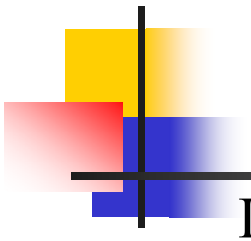
```
alloc( )
    head′ <= head
    head<=RAM[head]
return(head′)
```

- If an allocation and a free request occur in the same cycle, then the freed address can be used immediately to satisfy the allocation request, and the linked list remains unchanged.

FIFO Transformations



- When a portion of a thread is inactive, i.e it is not being used, it may sometimes make sense to store it in the RAM.
- The FIFO transformations, just like the route transformations, do not change the thread state.



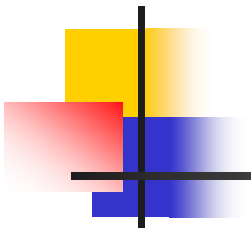
Flow Control and Scheduling

In general, some threads may attempt to use the same resources at the same time, causing deadlock, thus some sort of flow-control is necessary.

- One adequate approach is to use a simple control mechanism involving a valid bit and pause signal.
  - The valid bit defines whether the signal carries some valid data or not.
  - When a transformation such as merge cannot accept a thread, the pause signal is asserted.

There is no particular format to implement scheduling policies in the computational model. Scheduling policies depends on the system designer and also the components available.

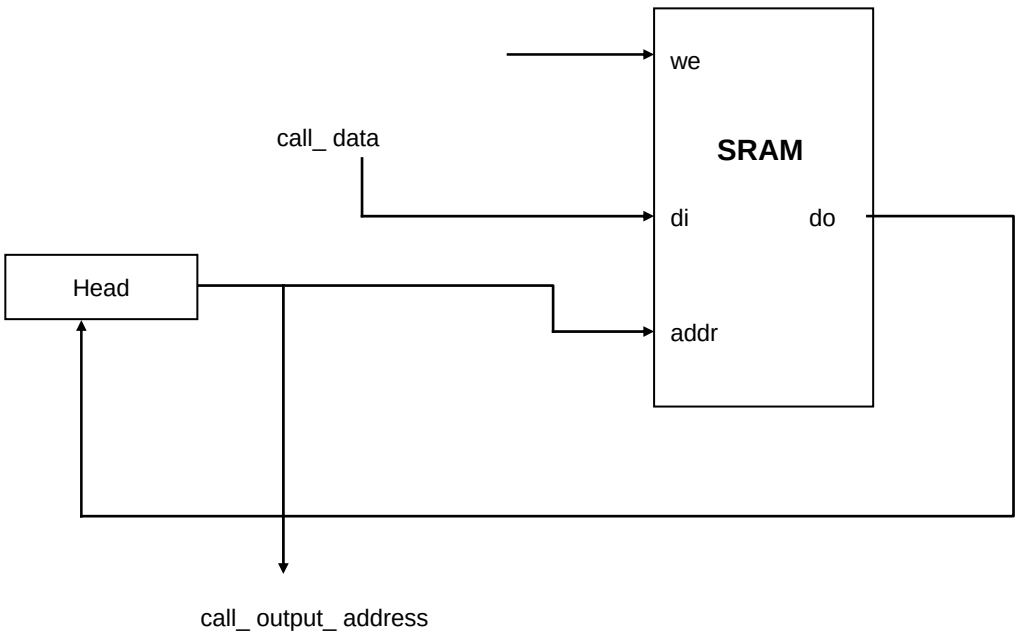
Scheduling policies for the Fibonacci program are discussed later.



Examples of Transformations

- Call-Return Block
- Non-Blocking Static Priority Router
- Send-Receive Block

Call-Return Block (Call Only)



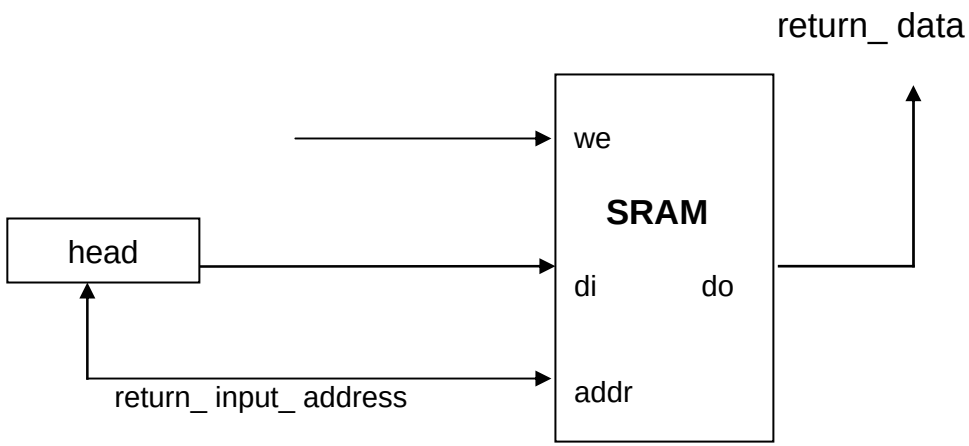
call_output_ address <= head

addr <= head

di <= call_data

head<= do

Call-Return Block (Return Only)



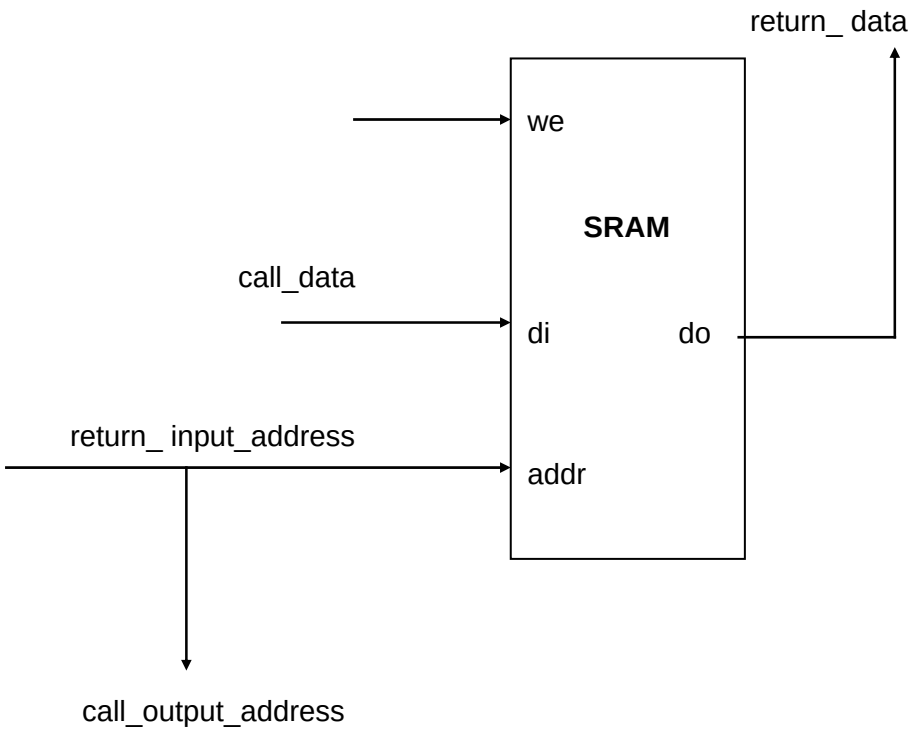
addr<= return_ input_address

return_data<= do

di<=head

head<= return_ input_address

Call-Return Block (Both Call and Return)



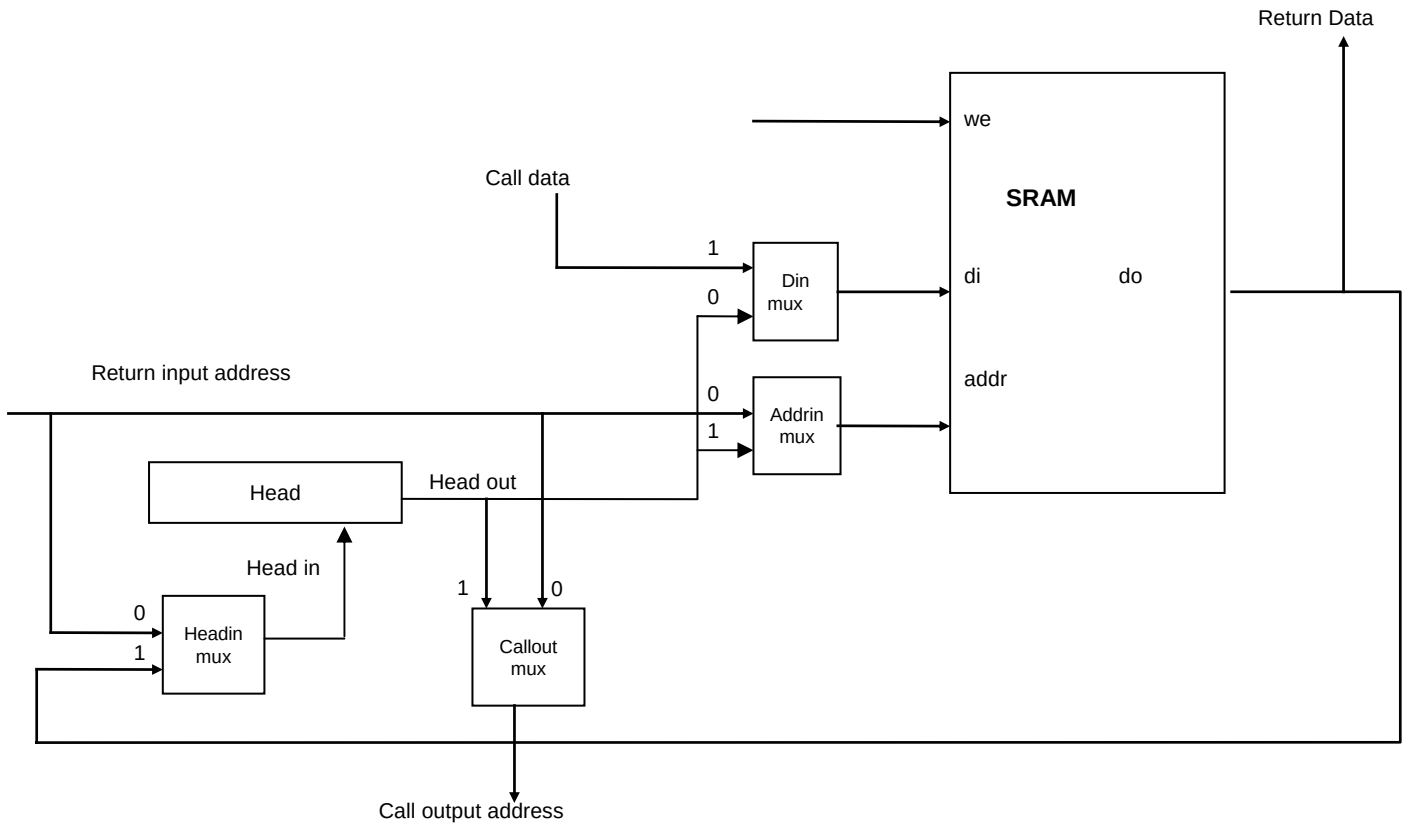
addr<= return_ input_address

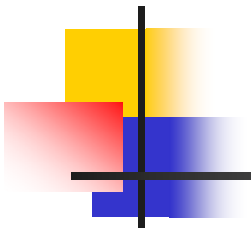
return_ data<= do

di <= call_data

call_output_address <=
return_input_address

Call-Return Block



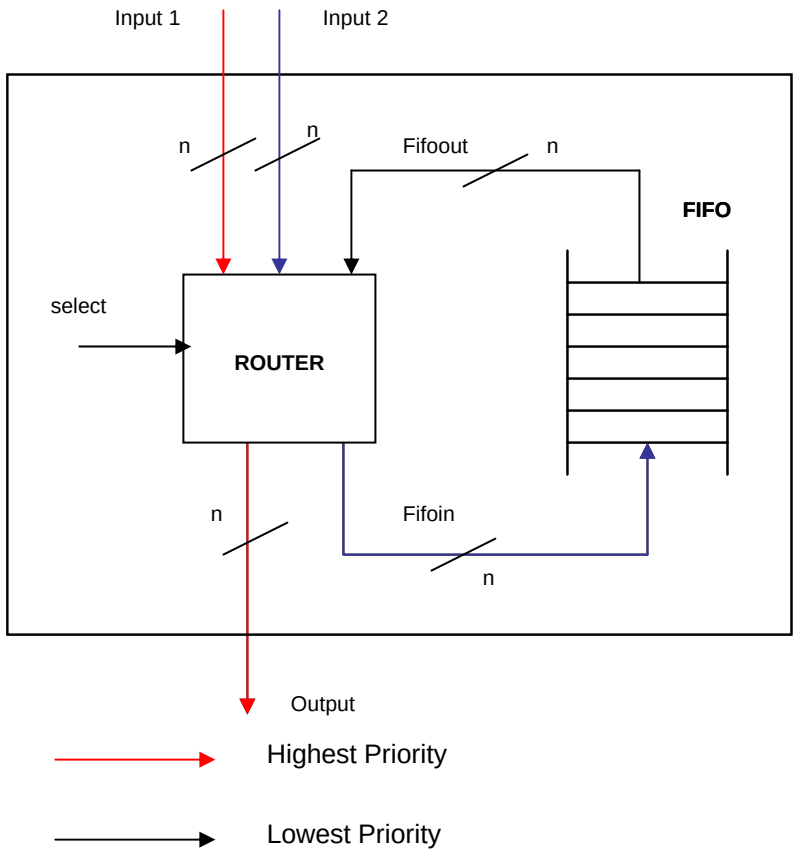


Scheduling Policies

Deadlocks are caused when more than one thread compete for the use of a transformation. Prudent use of FIFO and good capacity planning can be used to avoid deadlock.

- Round Robin strategy: fairness
- Blocking Static Priority Routers: only one thread is given priority and the other thread has to wait for the higher priority thread to complete.
- Non-Blocking Static Priority Router: higher priority thread is passed to the next transformation and the lower priority thread is placed in a FIFO

Non-Blocking Static Priority Router



VHDL Pseudo Code

```
case select is
     when "000" =>output<=(others=>'0');
     when "001" =>output<= FIFOOUT;
     when "010" =>output<= input2;
     when "011" =>output<= input2;
     when "100" =>output<= input1;
     when "101" =>output<= input1;
     when "110" =>output<= input1;
                    FIFOIN<=input2;
     when "111" =>output<= input1;
                     FIFOIN<=input2;
     when others=>NULL;
end case;
```

Example of Dual Transformations: Send-Receive Block

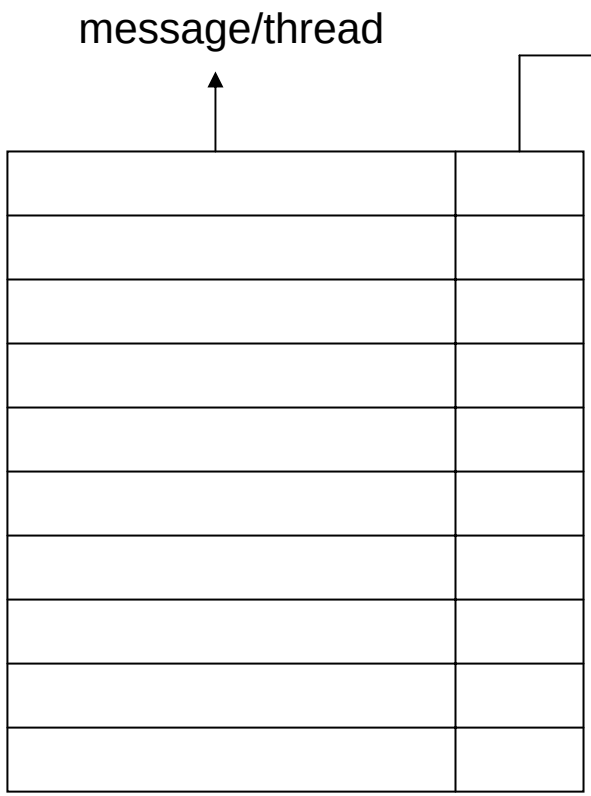
- Send-Receive module for interprocess communication allows threads to communicate among themselves without sharing data.
- With indirect communication, the messages are sent to and received from mailboxes.
- Each mailbox has a unique identification.
- The `Send` and `Receive` primitive are defined as follows:

  `Send` (A, `message`) – Send a message to mailbox A

  `message`=`Receive` (A) – Receive a message from mailbox A

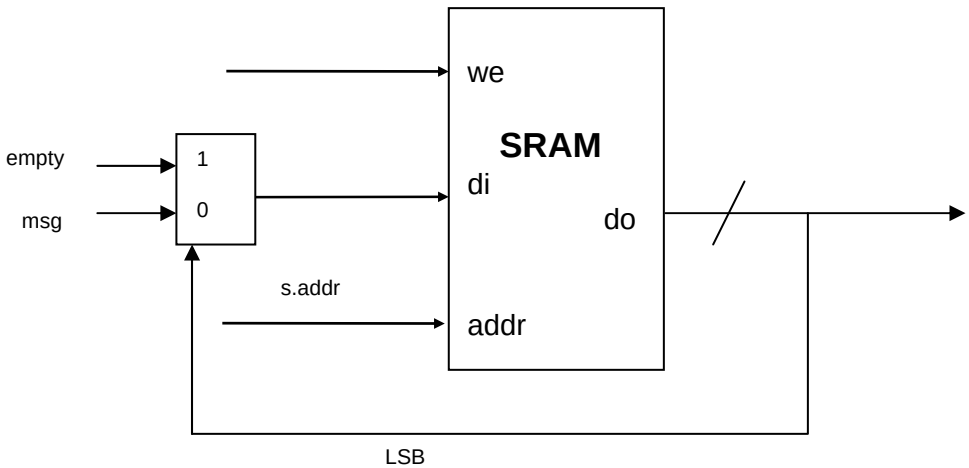
- When there is a message in the mailbox and a sender tries to send a new message to that mailbox then the old message is overwritten with the new one.
- When a receiver a tries to access a mailbox and there is no message for it then that thread blocks and waits for the message.

Status Bits



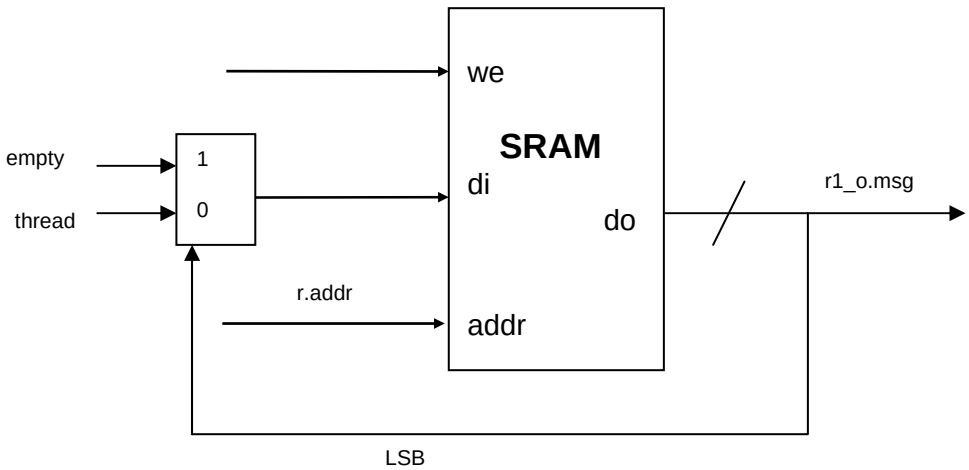
00 – Empty
01 – Message
10 – X
11 - Thread

Send-Receive Block (Send Only)




```
send( s.msg, s.addr)
if ( M[s.addr].status == empty)
     then     M[s.addr] <= s.msg
     M[s.addr].status <= msg
else if ( M[s.addr].status == thread)
     then     r2_o.msg <= s.msg
     r2_o.data<= M[s.addr].data
     M[s.addr].status <= empty
else ERROR
```

Send-Receive Block (Receive Only)

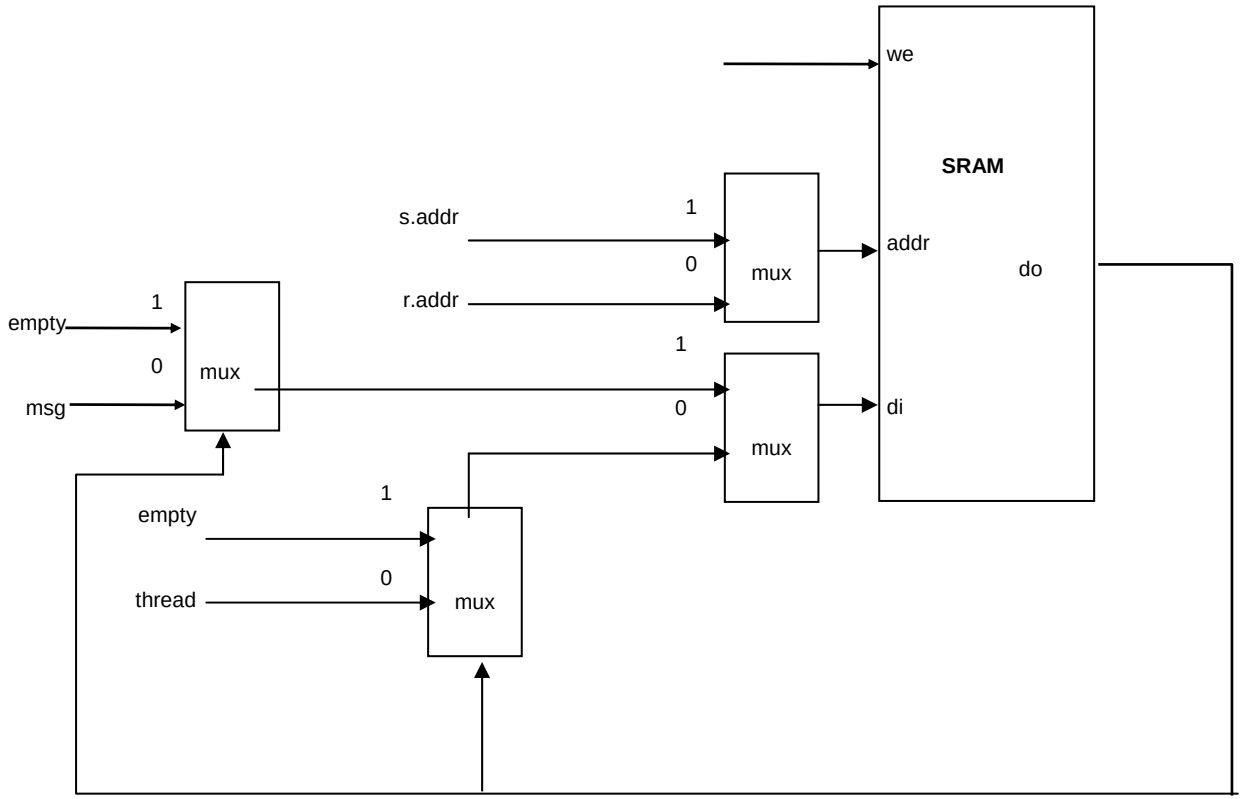



```
receive(r.thread, r.addr)
if (M[r.addr].status == empty) then
    M[r.addr] <= r.thread
    M[r.addr].status <= thread
else if (M[r.addr].status == msg)
then
    r1_o.msg<= M[r.addr].msg
    M[r.addr].status <= empty
else ERROR
```

Send-Receive Block (Both Send and Receive)

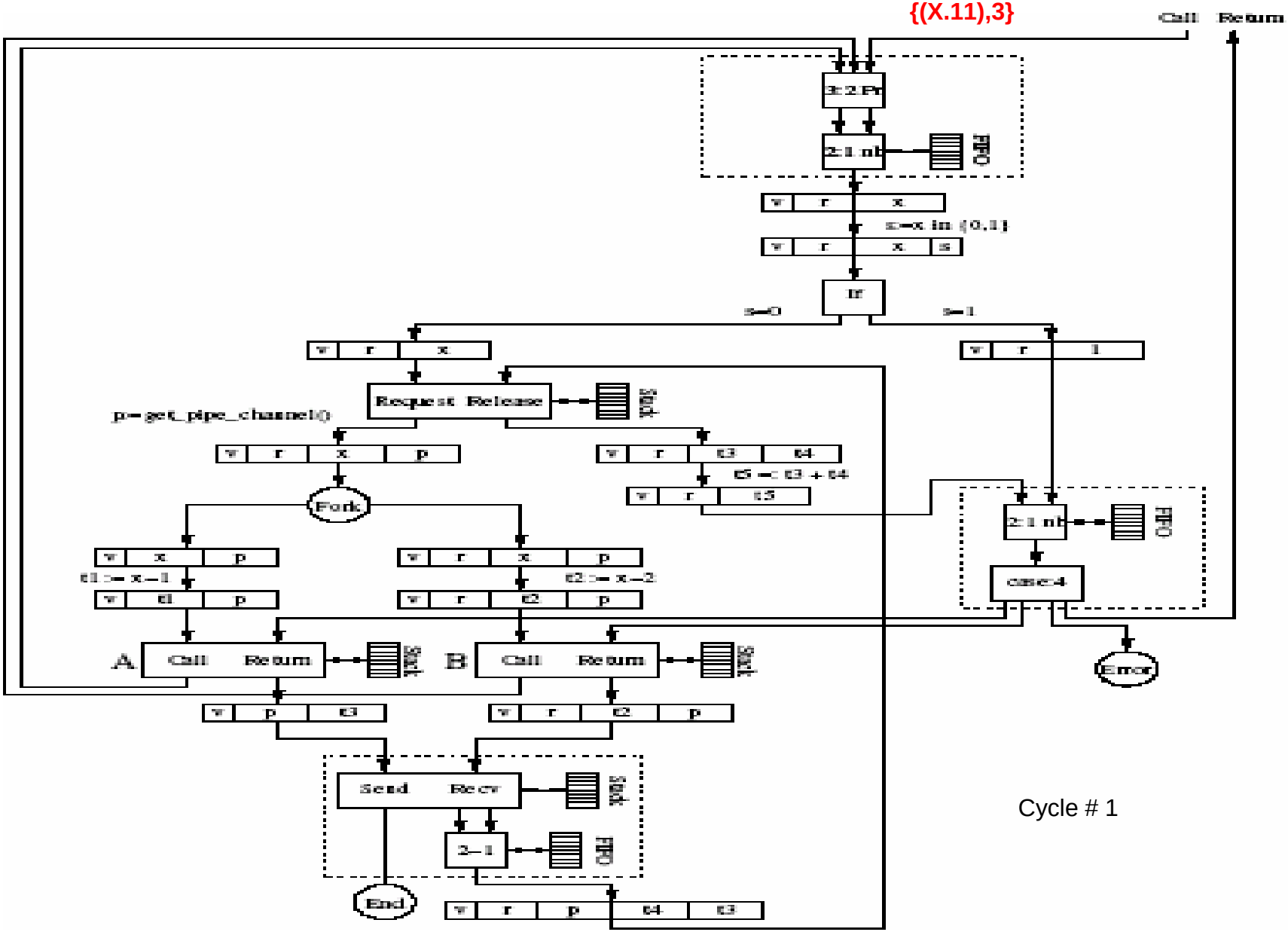
```
send( s.msg, s.addr), receive(r.thread, r.addr)
        if ( s.addr == r.addr) then
            r1_o.msg <= s.msg
            r1_o.data<= r.thread
```

Send-Receive Block



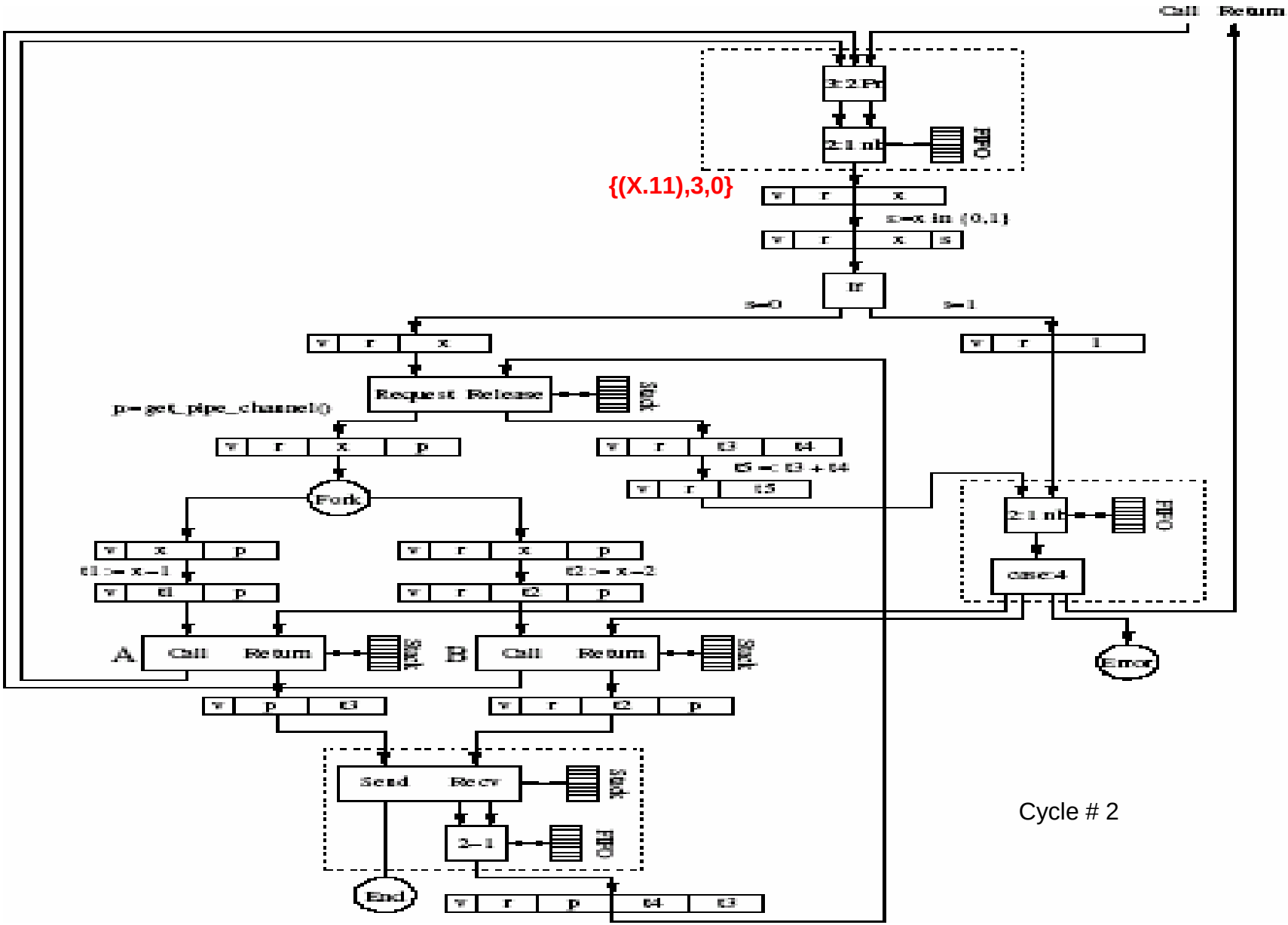
Model of Computation Fibonacci

{(X.11),3}

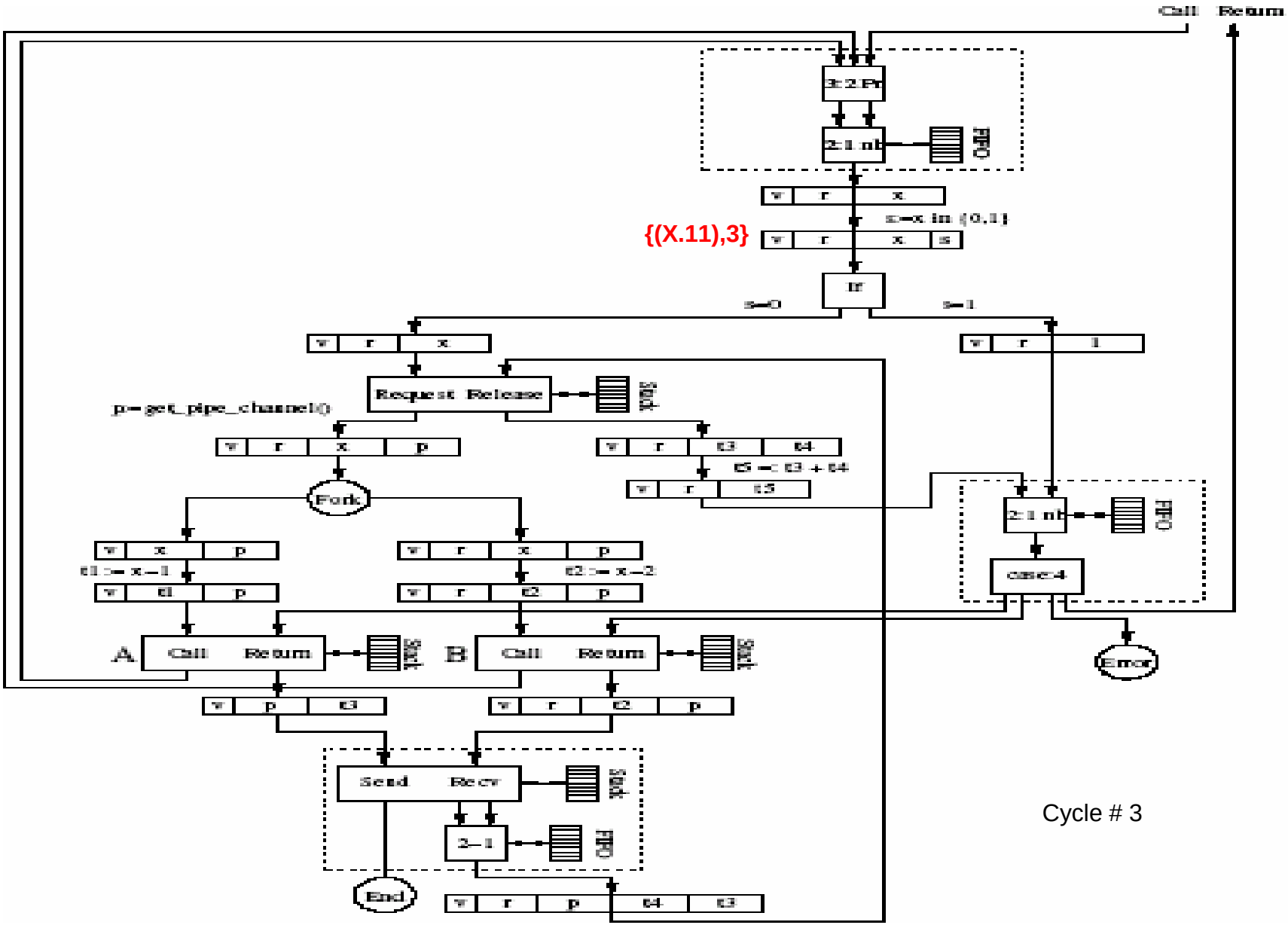


Cycle # 1

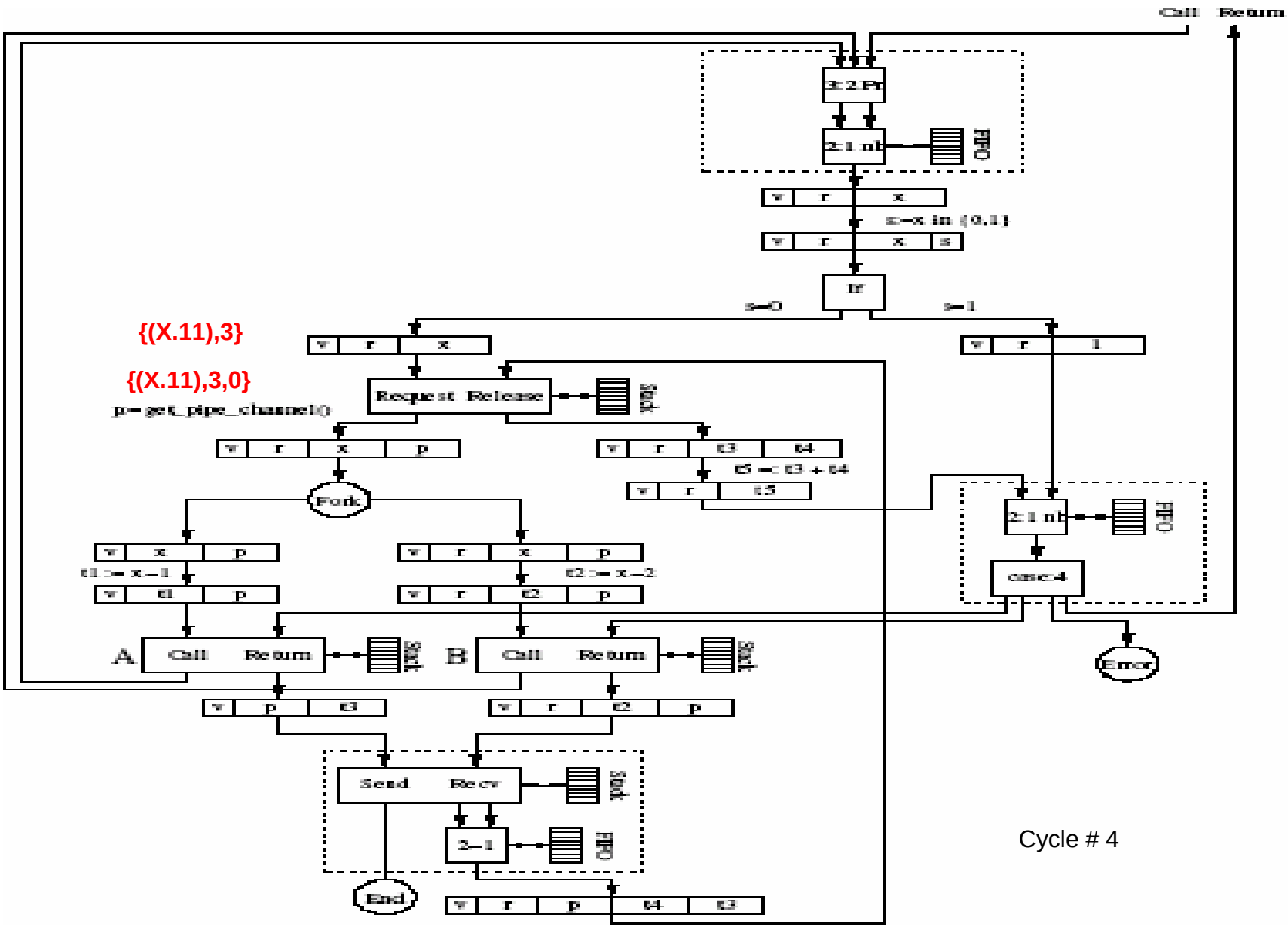
Model of Computation Fibonacci



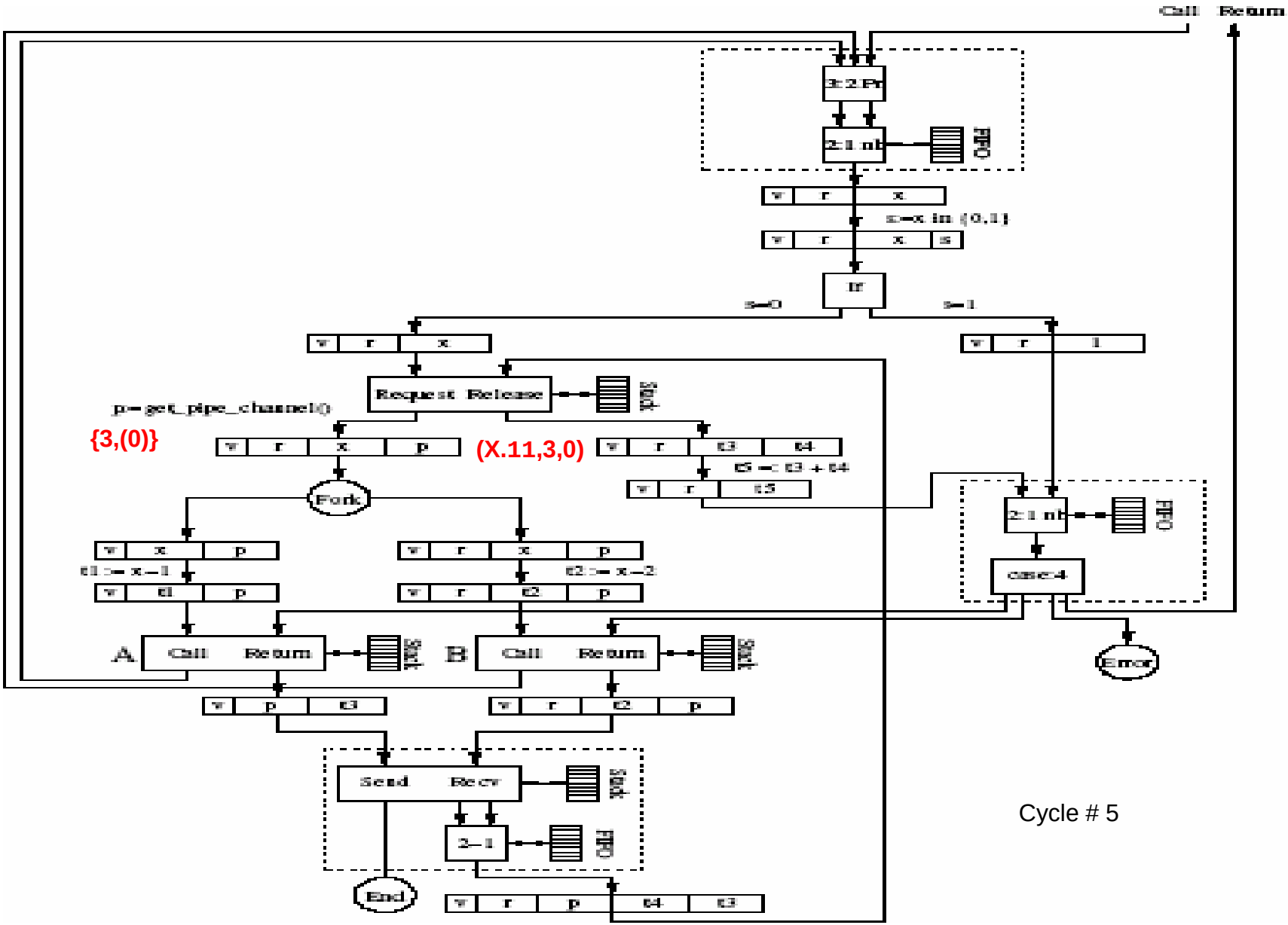
Model of Computation Fibonacci



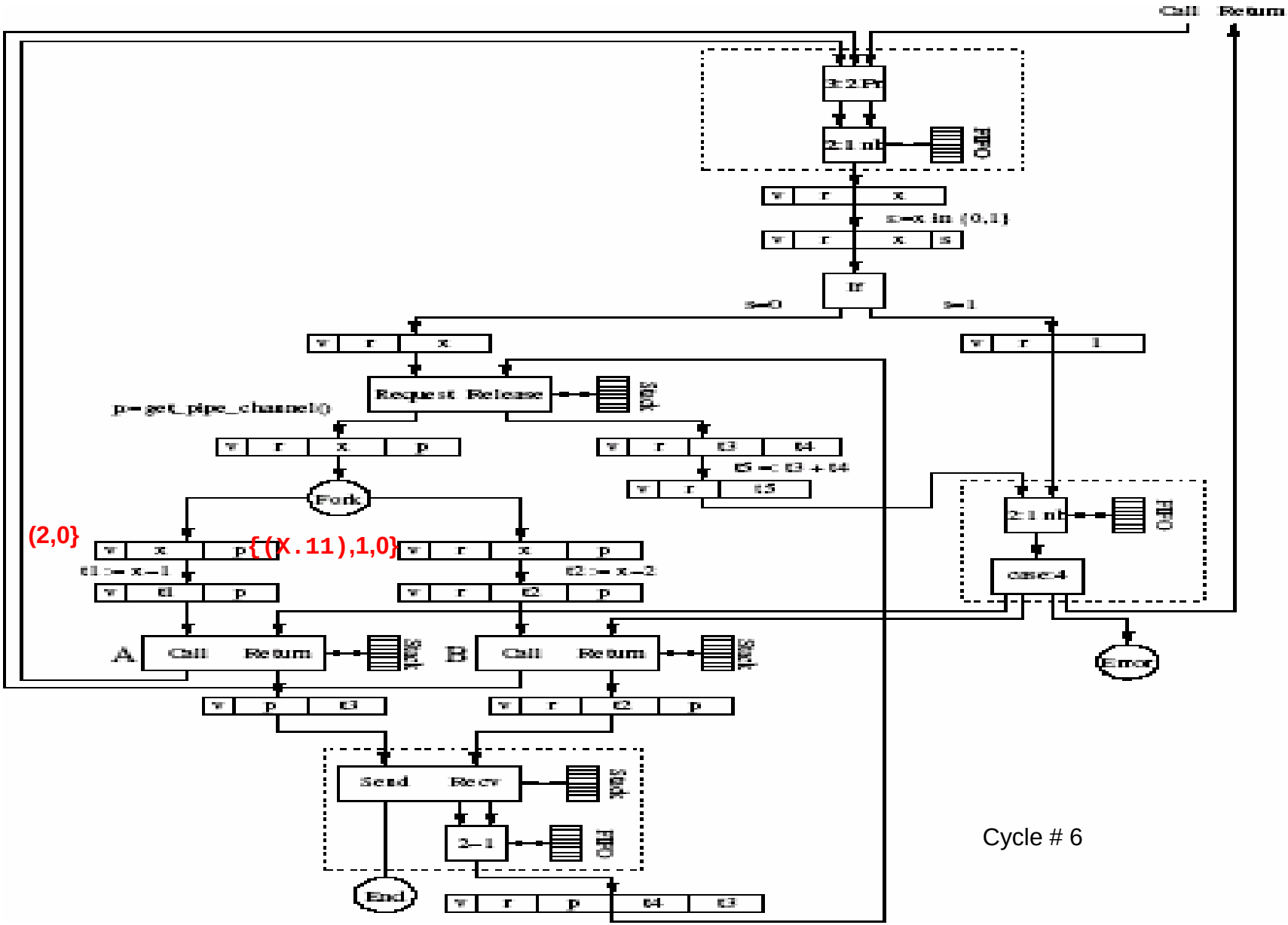
Model of Computation Fibonacci



Model of Computation Fibonacci

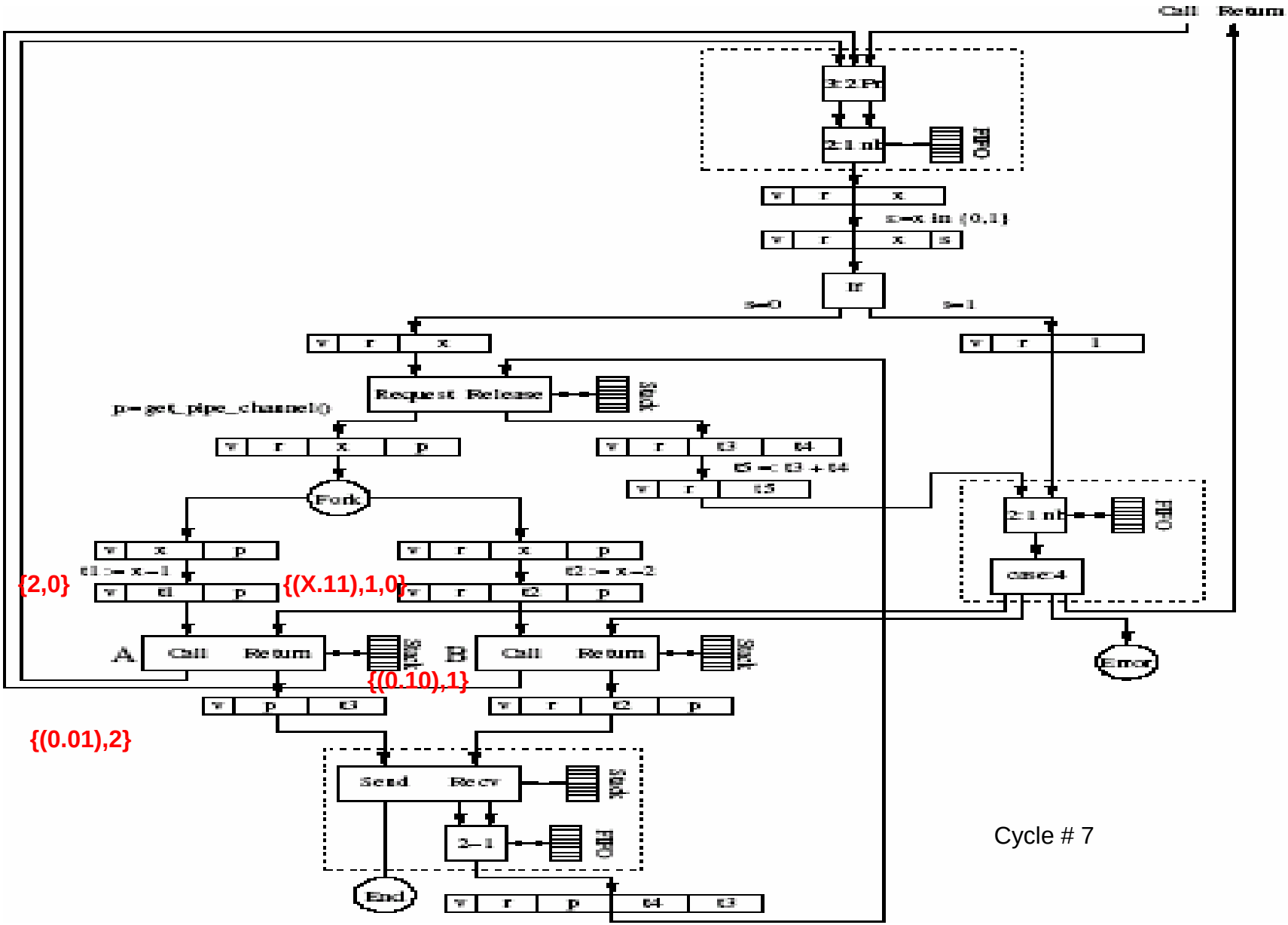


Model of Computation Fibonacci



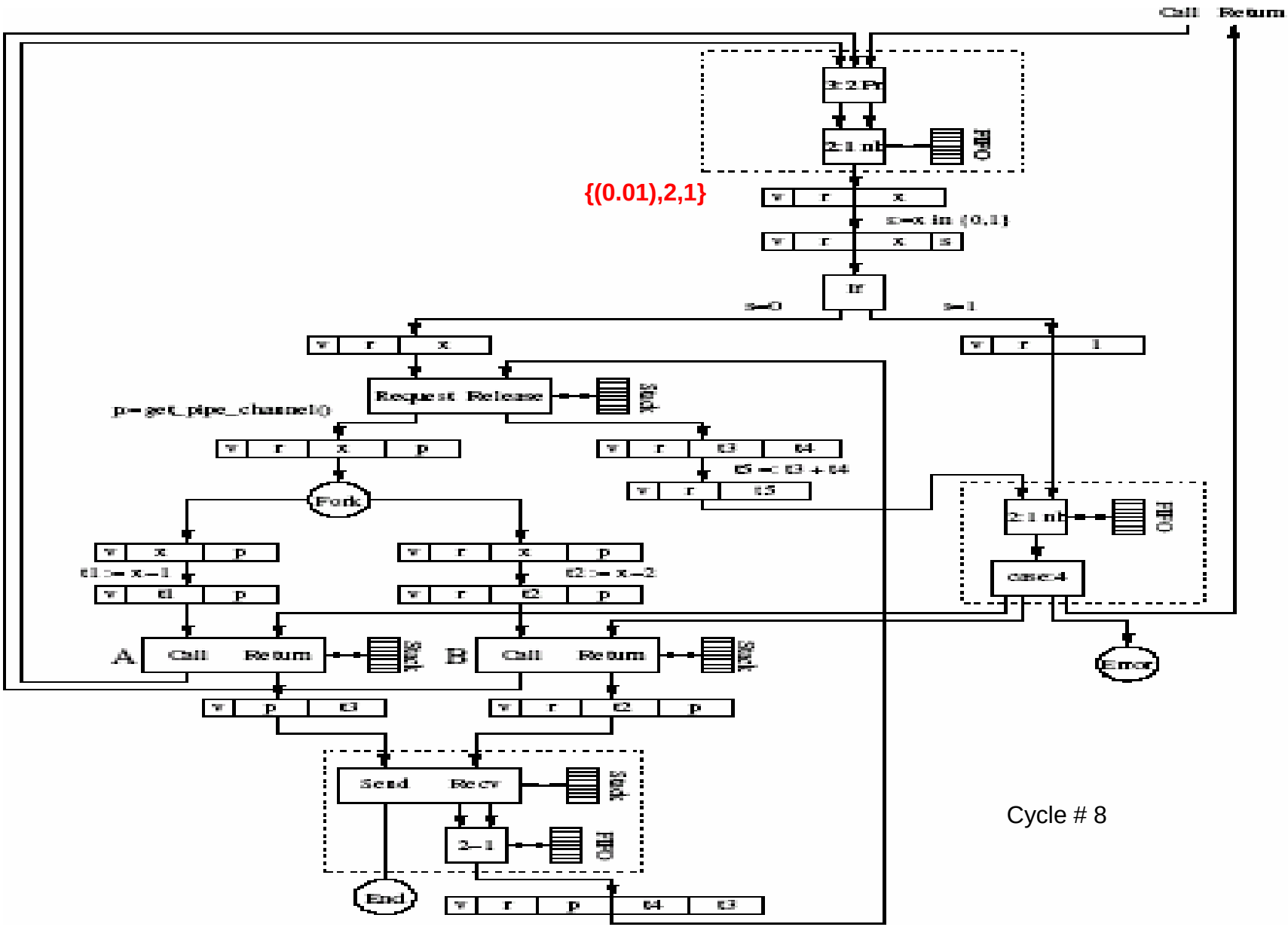
Cycle # 6

Model of Computation Fibonacci

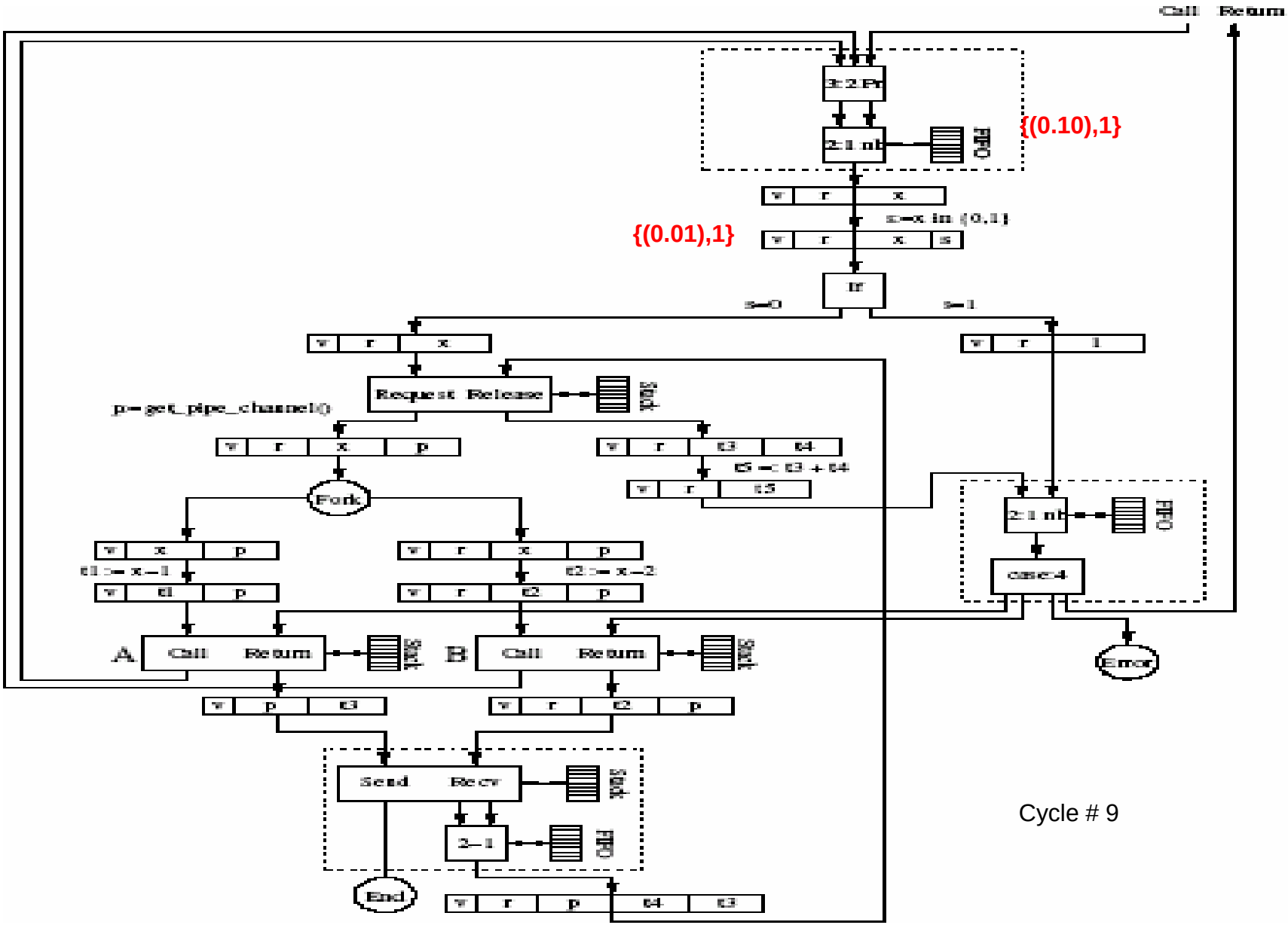


Cycle # 7

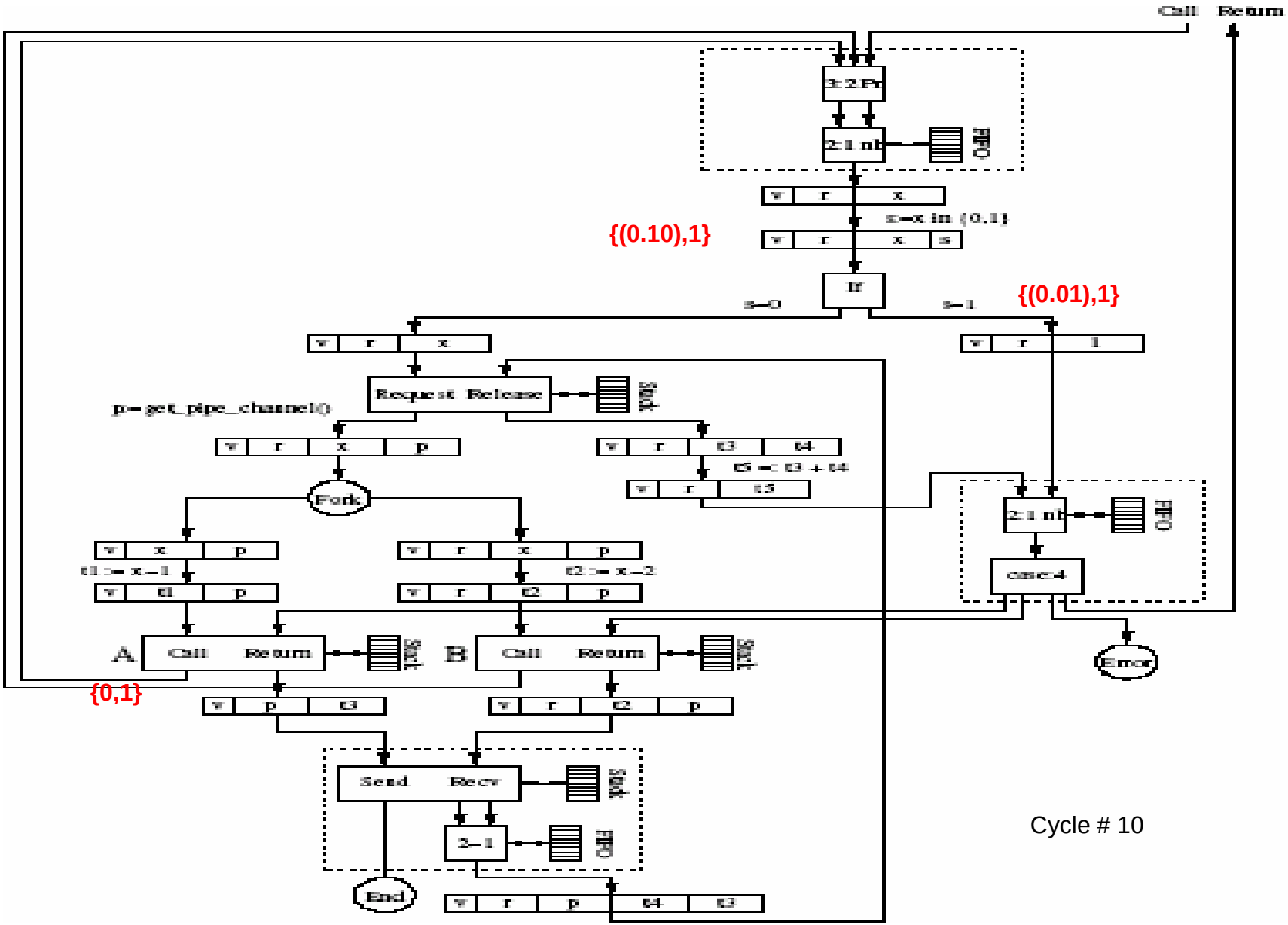
Model of Computation Fibonacci



Model of Computation Fibonacci

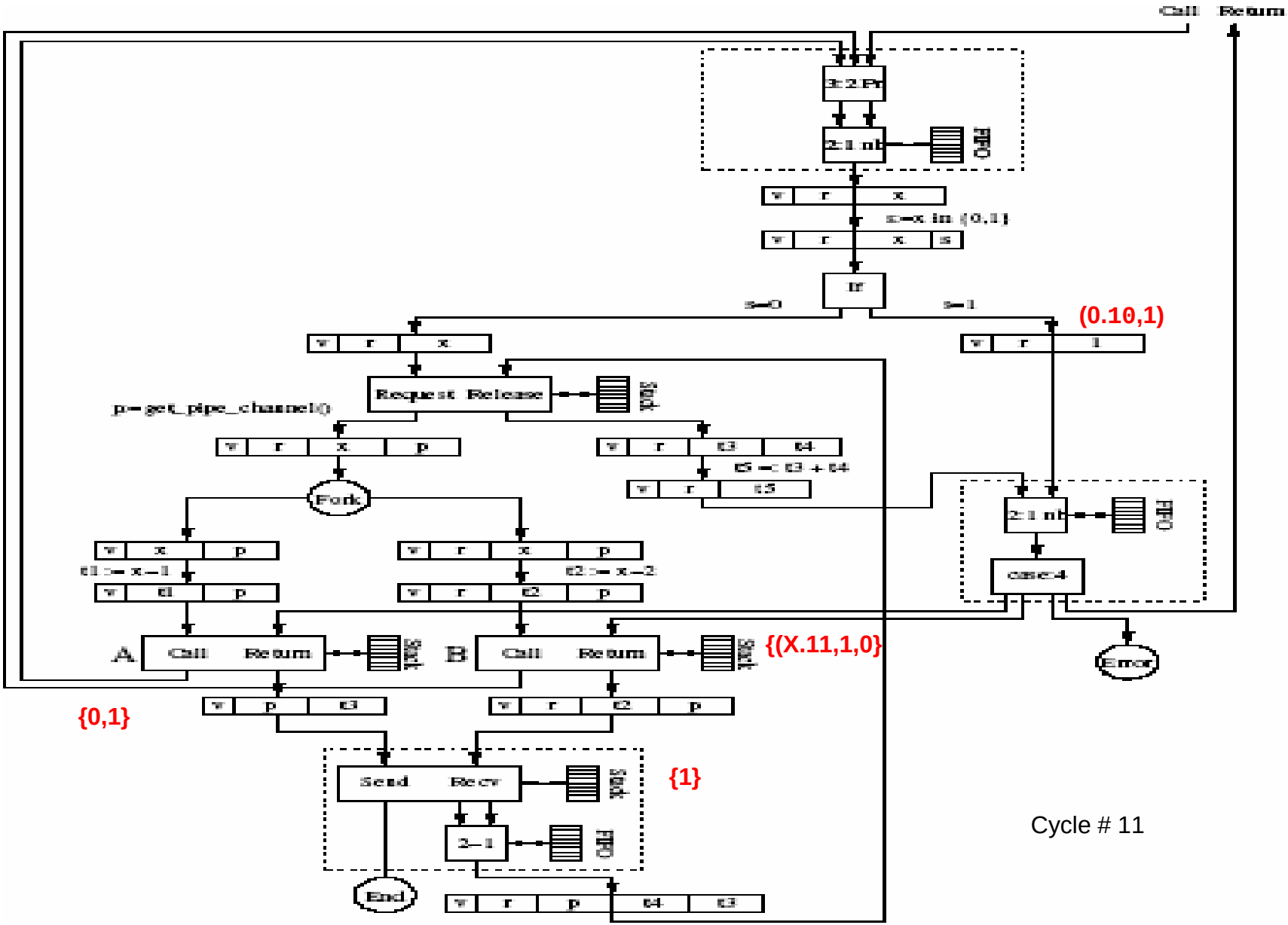


Model of Computation Fibonacci



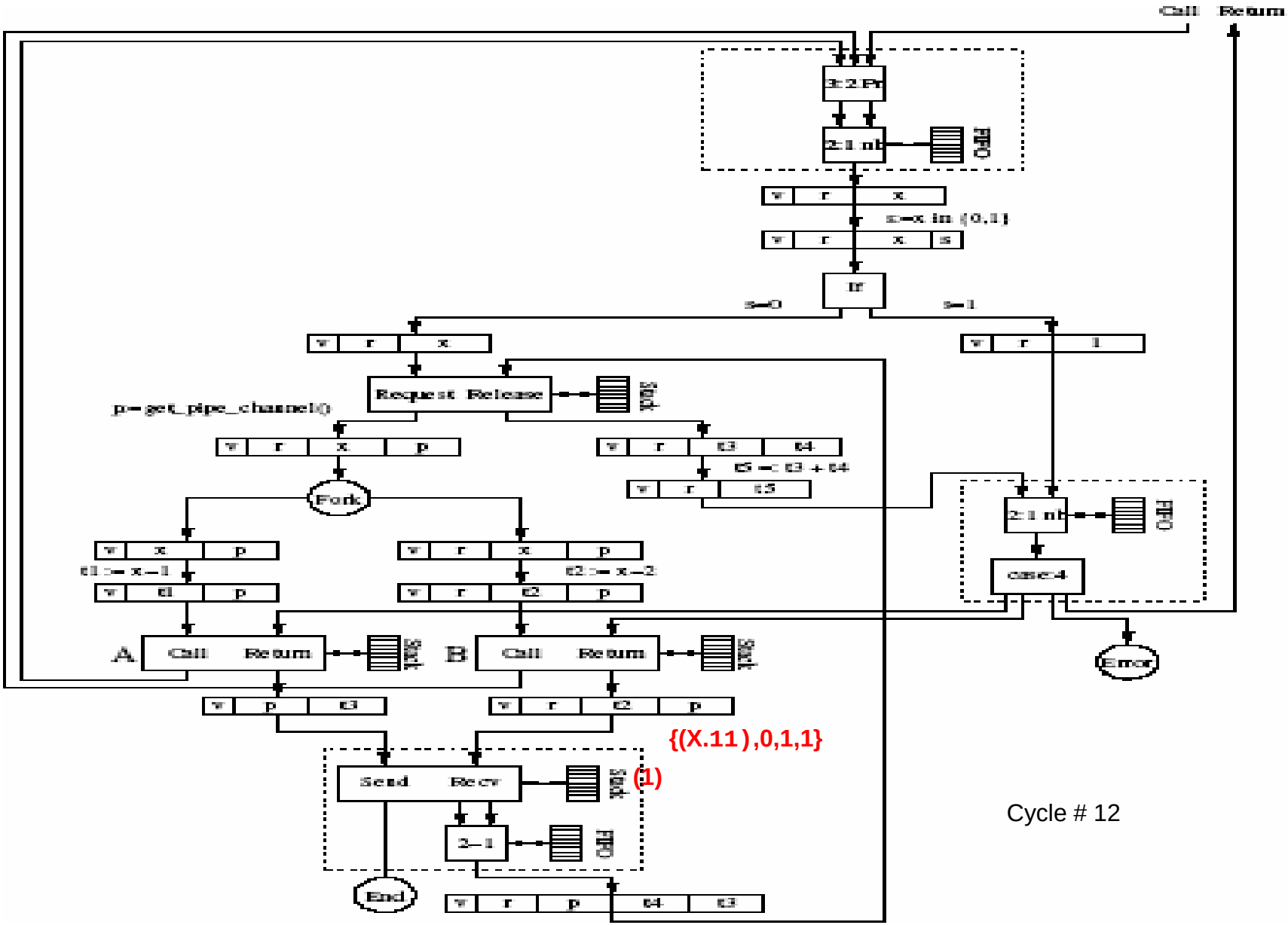
Cycle # 10

Model of Computation Fibonacci

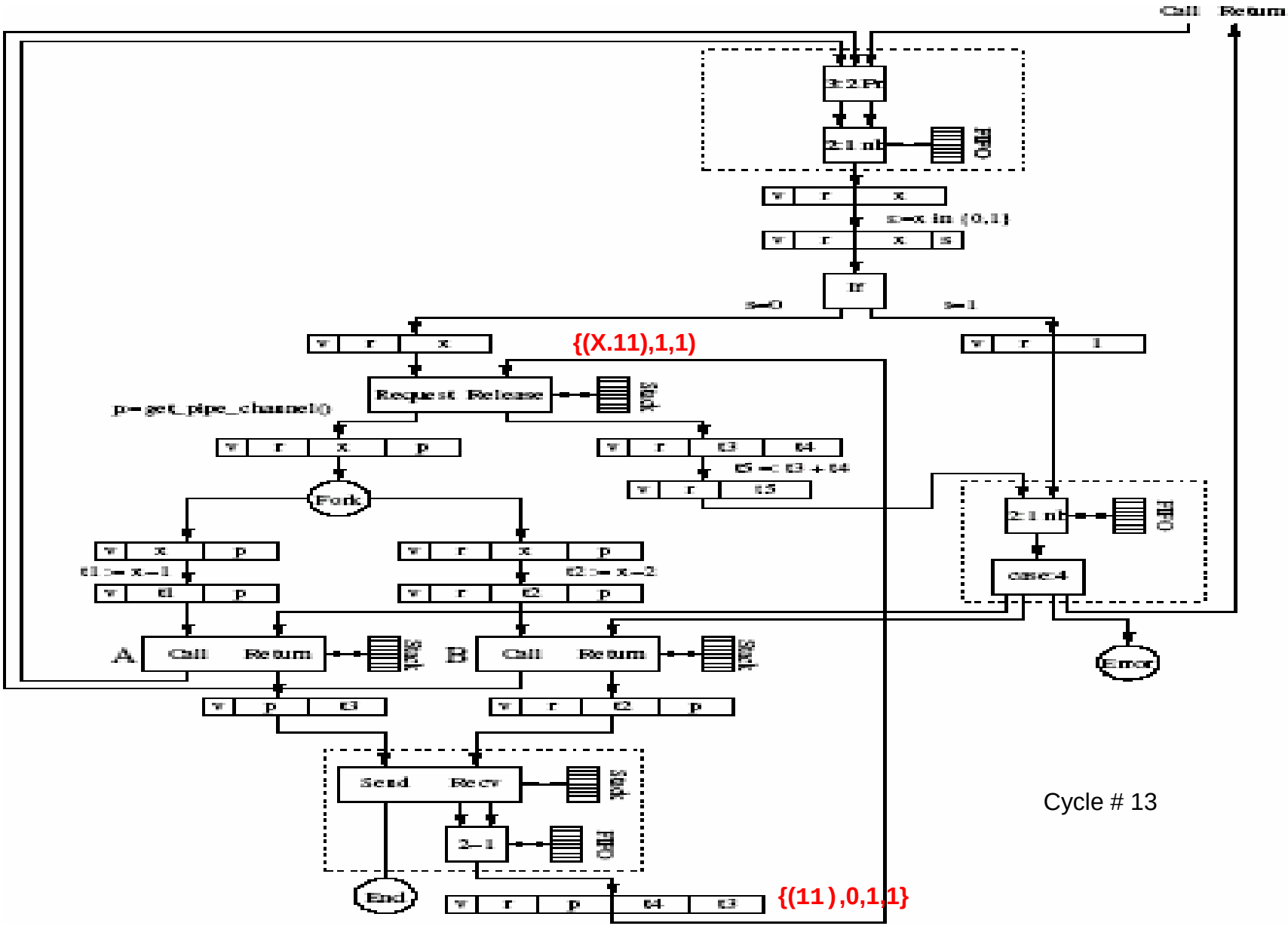


Cycle # 11

Model of Computation Fibonacci

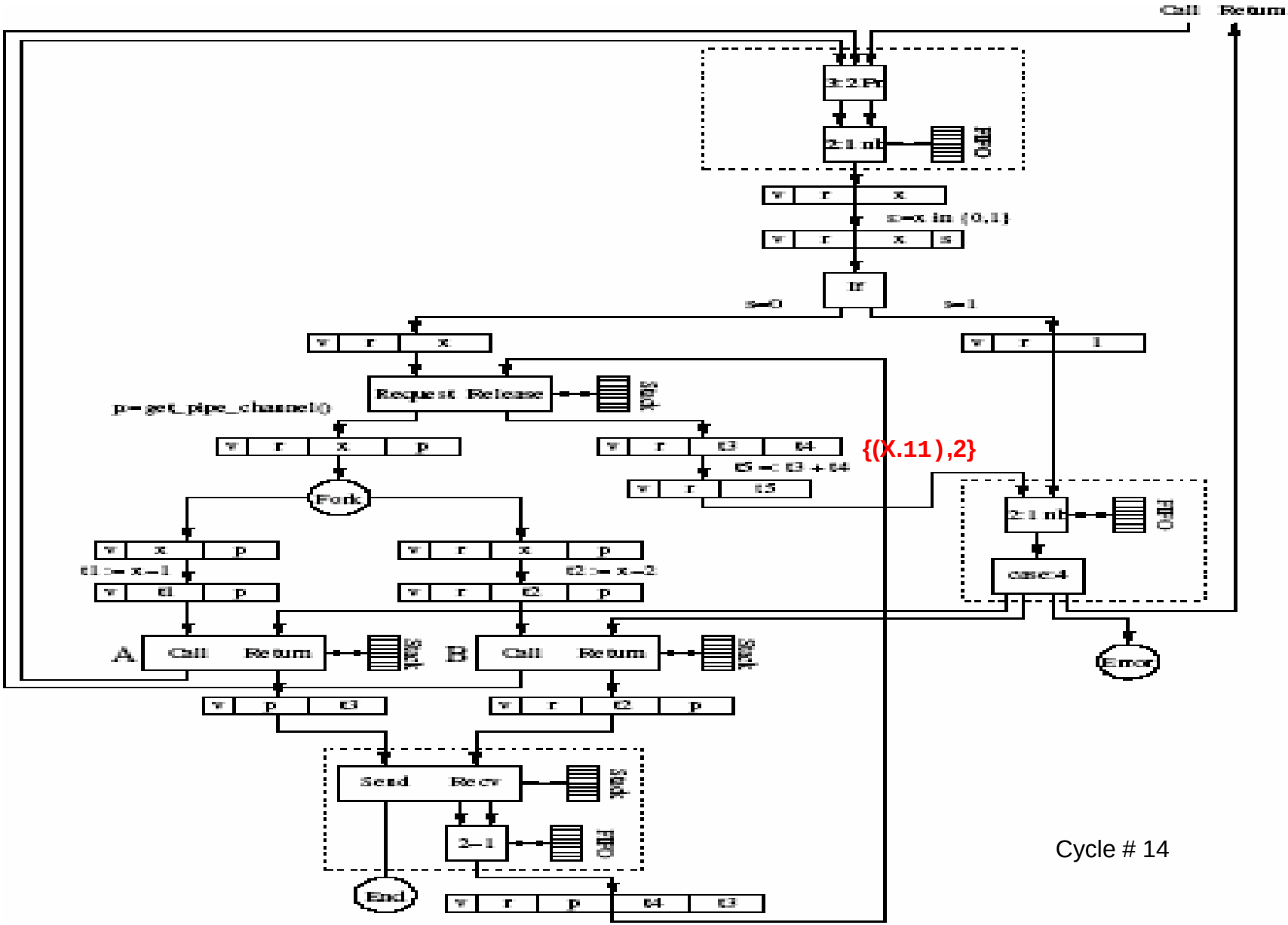


Model of Computation Fibonacci



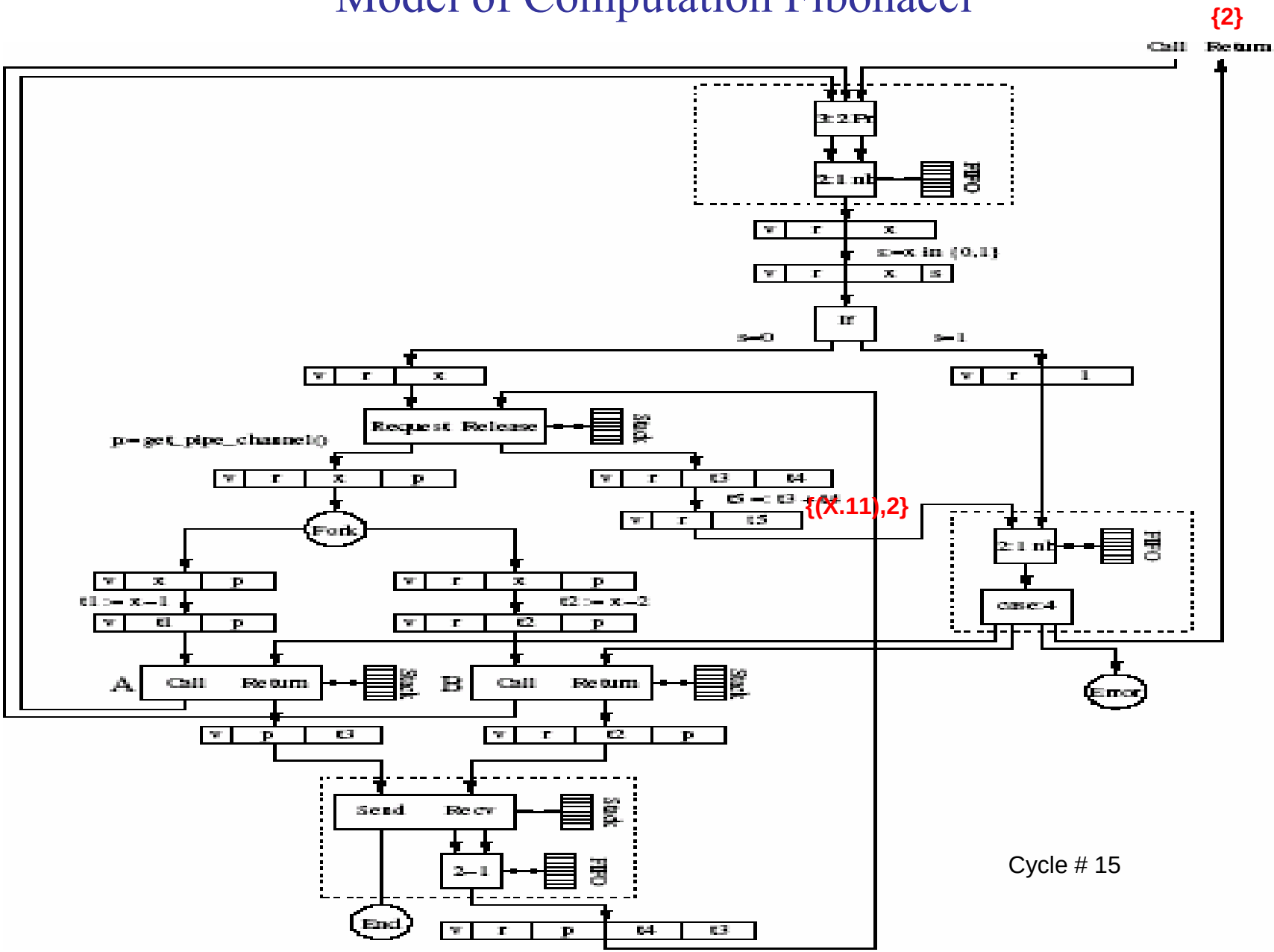
Cycle # 13

Model of Computation Fibonacci



Cycle # 14

Model of Computation Fibonacci



Cycle # 15

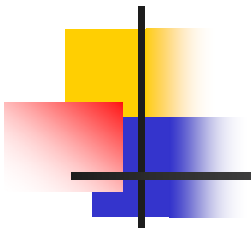
Fibonacci Results

Simulation Results

Input	Input Cycle Number	Output	Output Cycle Number
8	2	21	103
1,2	2,3	1,1	3,4
1,3,**5**	2,3,**4**	1,2,**5**	3,26,**41**
1,2,4	2,3,4	1,1,3	3,4,29
1,2,**5**,6	2,3,**4**,5	1,1,**5**,8	3,4,**61**,63
1,2,3,4,**5**	2,3,4,5,**6**	1,1,2,3,**5**	3,4,25,36,**50**
1,2,3,4,**5**,7	2,3,4,5,**6**,7	1,1,2,3,**5**,13	3,4,30,55,**90**,124

Observations:

- The number of cycles taken to obtain the fibonacci of a number is dependent on the number of threads.



Fibonacci Results

Input	Threads	Calls	Mailboxes	Cycles
1	1	0	0	3
2	1	0	0	3
3	3	2	1	15
4	5	4	2	25
5	9	8	4	36
6	15	14	7	48
7	25	24	12	75
8	41	40	20	103

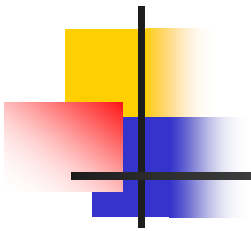
Fibonacci Results

Synthesis Report Summary

Device utilization summary

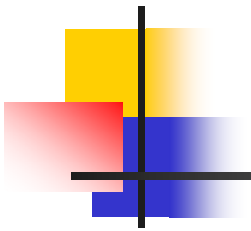
Selected Device: 2vp20ff1152-7

Number of Slices:	1088 out of	9280	11%
Number of Slice Flip Flops:	346 out of	18560	1%
Number of 4 input LUTs:	1762 out of	18560	9%
Number of bonded IOBs:	12 out of	564	2%
Number of GCLKs:	2 out of	16	12%



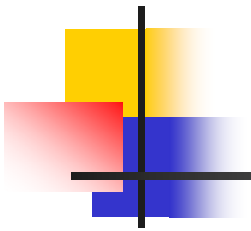
Future Work

- BlockRAM instead of Distributed SelectRAM
- The programs were limited to input sizes that were small. Increase in the input size would increase the number of resources used
- The next step to improve this computational model will be to implement pointers to functions and to include memory management capabilities.
- Integrating this work with hybrid threads



Features

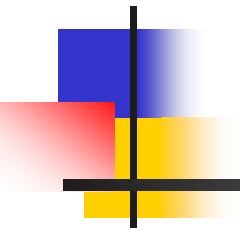
- Fully recursive
- High-level Concurrency, allows multiple threads
- Implements complex constructs such as call-return subroutine and message passing
- Utilizes modest resources



Conclusion

- There is now a computational model that allows reconfigurable logic to provide an excellent base for the design and implementation of various complex algorithms such as genetic algorithms in hardware.

- This computational model will help the system designers to bridge the gap between hardware and software.



Thank You