2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

A Reconfigurable Interface Architecture to Protect System IP

Dr. Arshad Riazuddin and Dr. Shoab A. Khan

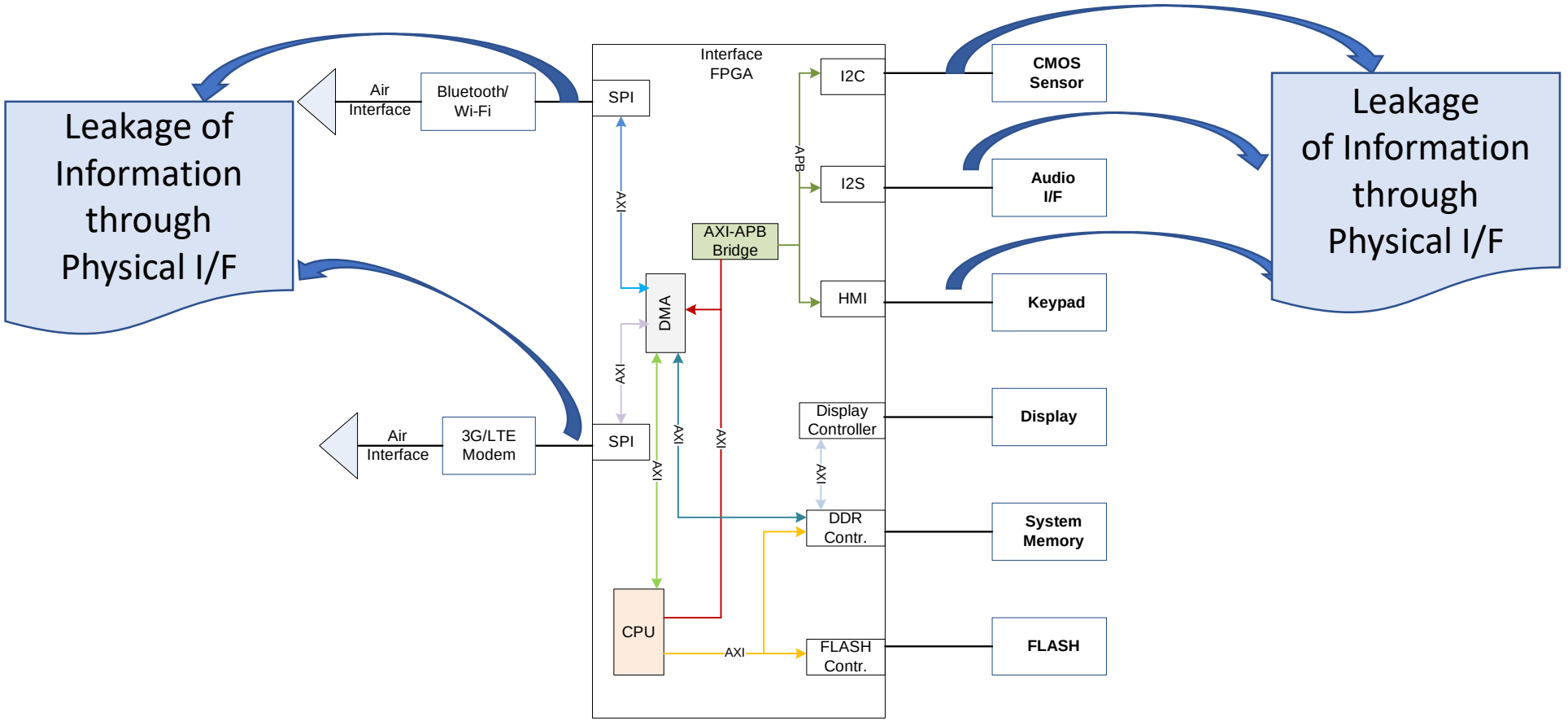
Center for Advanced Research in Engineering

accellera SYSTEMS INITIATIVE

Problem

- Information leakage through physical interface
- Such an attack can be intelligent or brute force
- This information can be used to decipher the Intellectual Property (IP) of the system
- How to solve this problem?

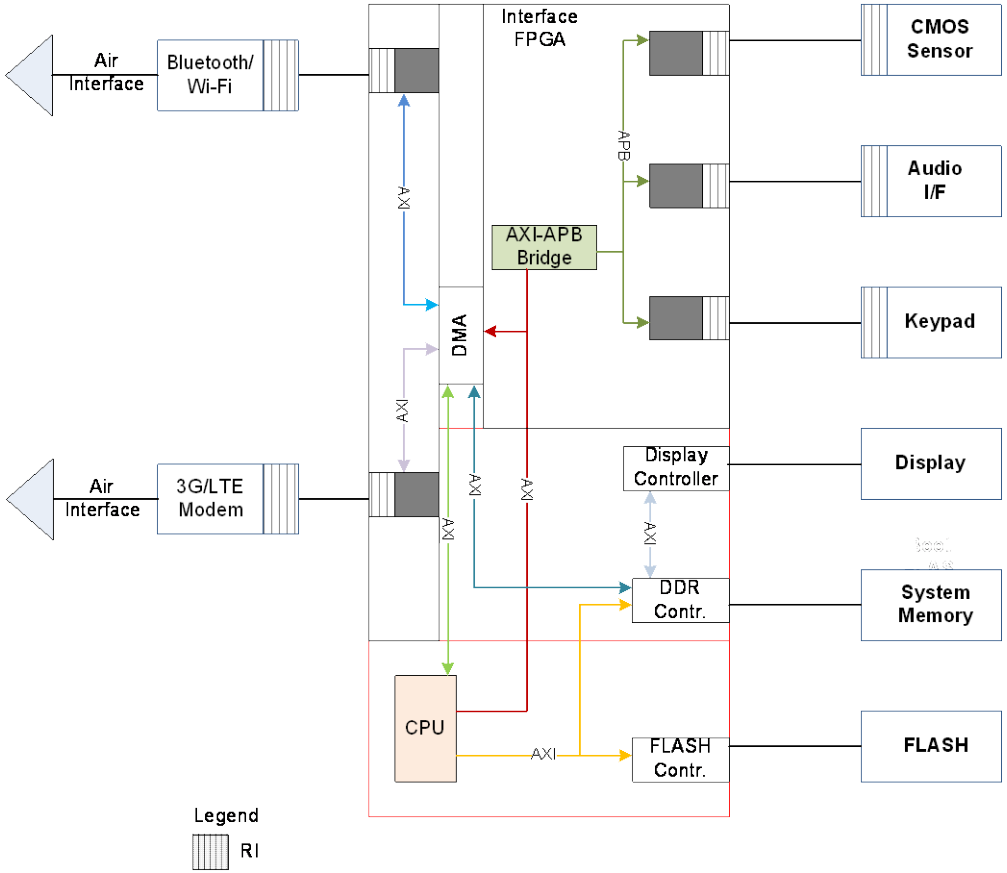
Example System



Proposed Solution

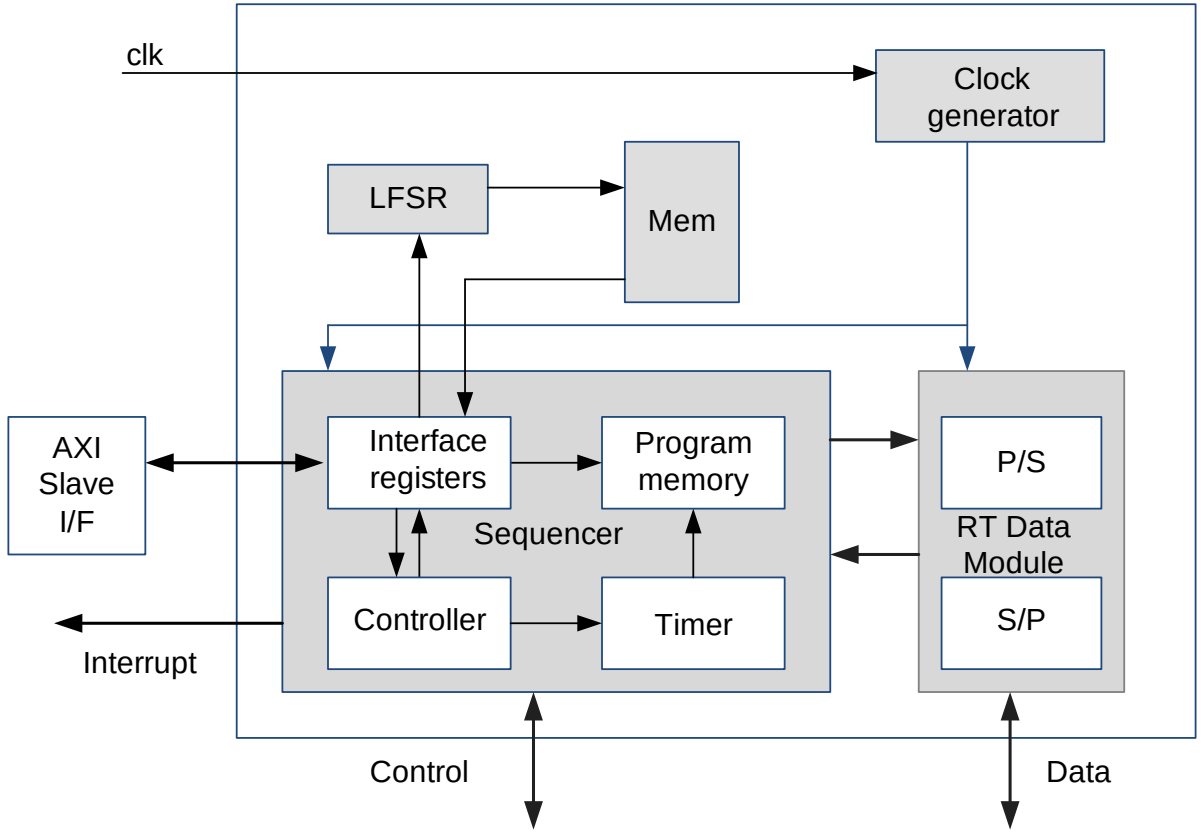
- How do we protect IP of a system?
  - Make the interconnect behavior change over time
- An “on the fly” reconfigurable interface
- The architecture allows
  - Change of interface architecture
  - Interface can be changed at a variable rate
- Supports Standards and Proprietary Interfaces
- Independent of any ASIC/FPGA architecture

System with Reconfigurable Interface (RI)



Reconfigurable Interface Architecture

- Reconfigurable Interface
  - Programmable Sequencer
  - R/T Data Module
  - Variable Interface Change



Sequencer Architecture

- Sequencer consists of
  - Controller
  - Program Memory
  - Timer
  - Interface Registers

Sequencer OPCODES

Opcode	Description
No Operation (NOP)	No Operation
Sample Flag (SAMPF)	Wait for an input flag to be equal to a particular value, before going to next instruction
Compare Flag (CMPF)	Compare the flag, before going to next instruction
Jump Unconditionally (JUMP)	Jump unconditionally to the address specified in the control word
Jump Conditionally (JUMPC)	Jump conditionally to the address specified in the control word

Sequencer Control Word

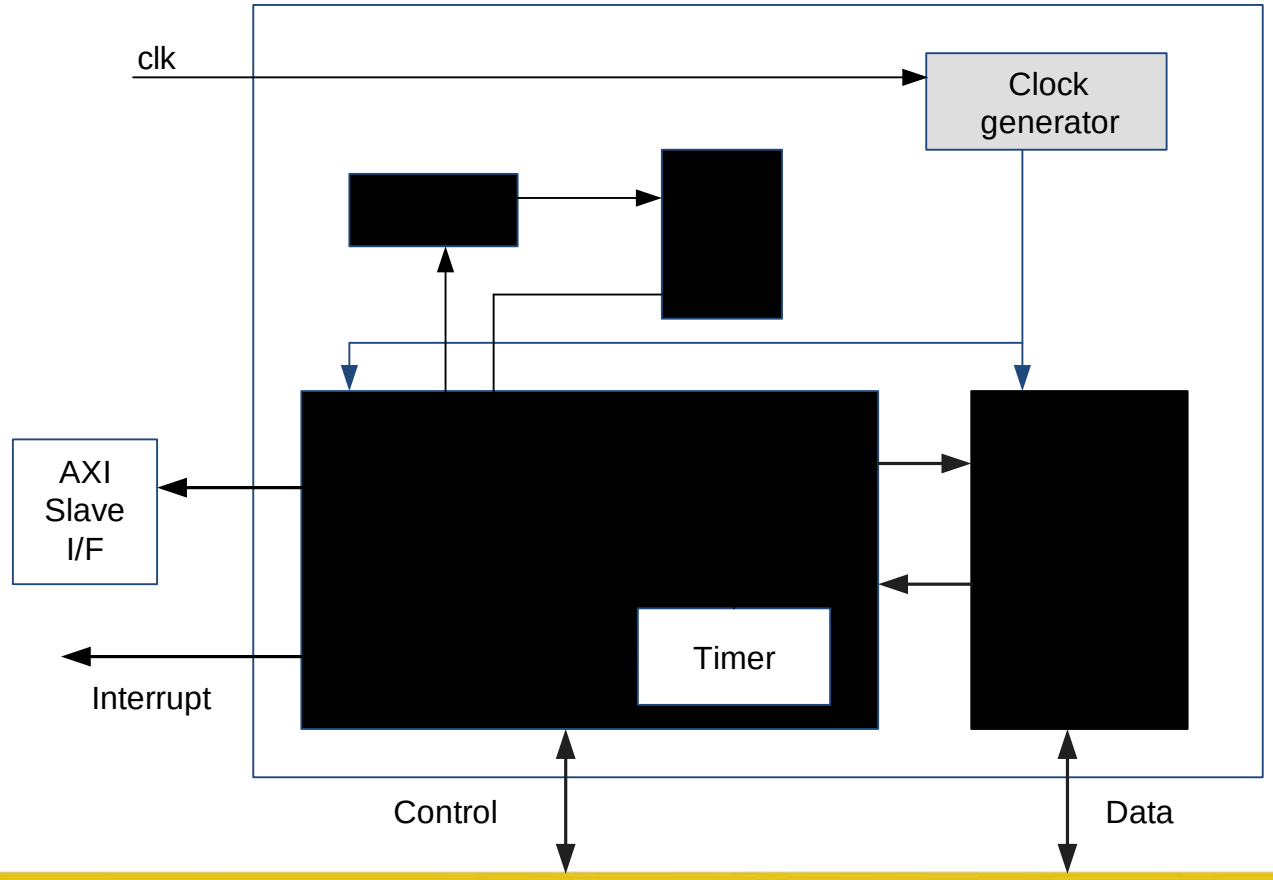
Bits	Description
31:29	Opcode
28:21	Jump address used in jump instruction
20:0	Wait for conditional flags which are used to move to the next instruction in the sequence

Description of Flags in Seqeuncer

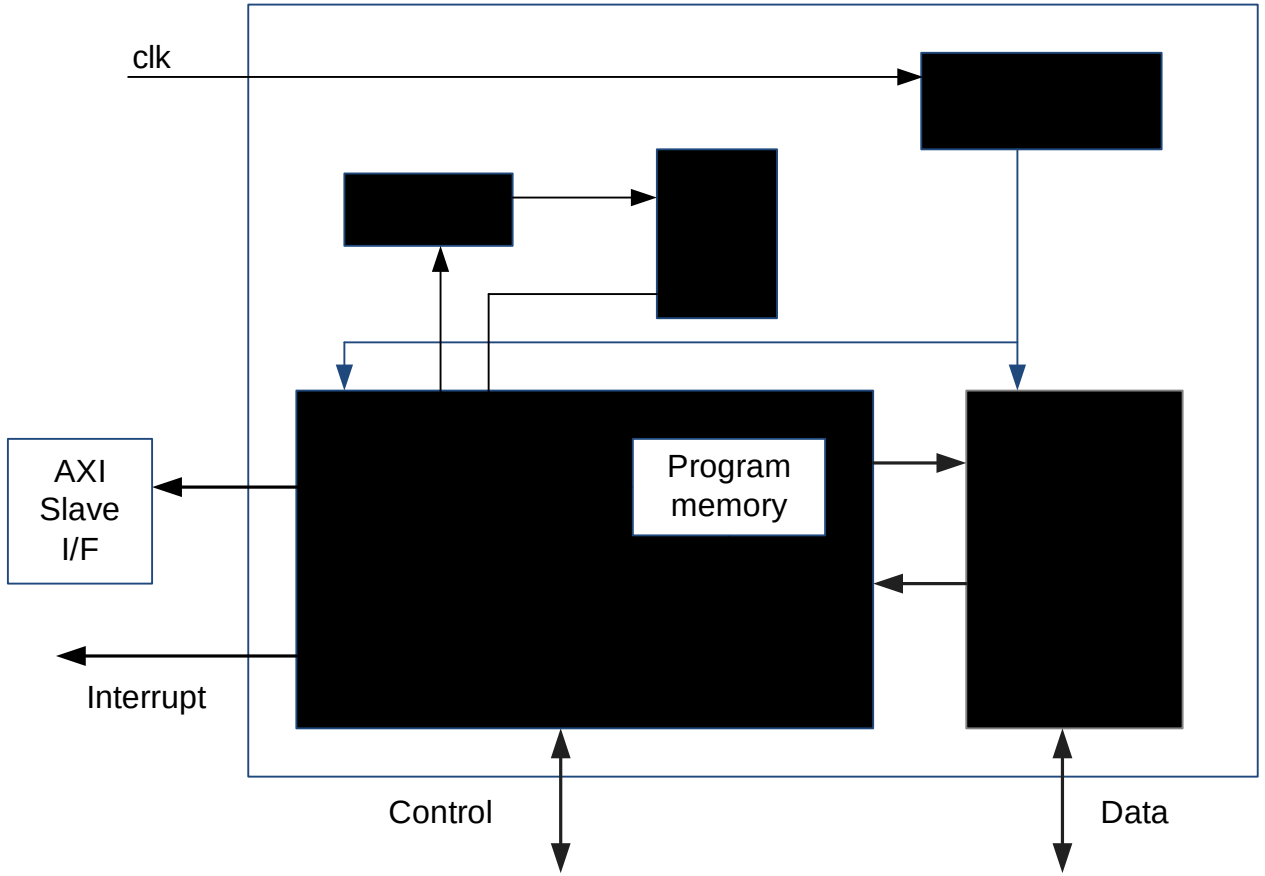
FLAG Description

Flag Bit	Direction of Flag	Description
F20	Input	Receiver Shift Enable
F19	Output	Sequencer Busy
F18	Output	Interrupt Request
F17	Output	Load Timer
F16	Input	Clock Pulse
F15	Output	Clock Generator Enable
F14	Output	Transmitter Enable
F13	Output	Receiver Enable
F12	Input	Terminal Counter Timer
F11	Input	Transmitter Shift Done
F10	Input	Receiver Shift Done
F09	Input	Transmitter Shift Enable
F08	Output	Serial Data Output Enable
F07	Output	Serial Data Input Enable
F06	Output	Serial Clock Output Enable
F05	Output	Serial Chip Select
F04	Input	Transmitter Shift Register Empty/Receiver Shift Register Full
F03	Input	Valid Start Bit
F02	Output	Interface Select Bit 2
F01	Output	Interface Select Bit 1
F00	Output	Interface Select Bit 0

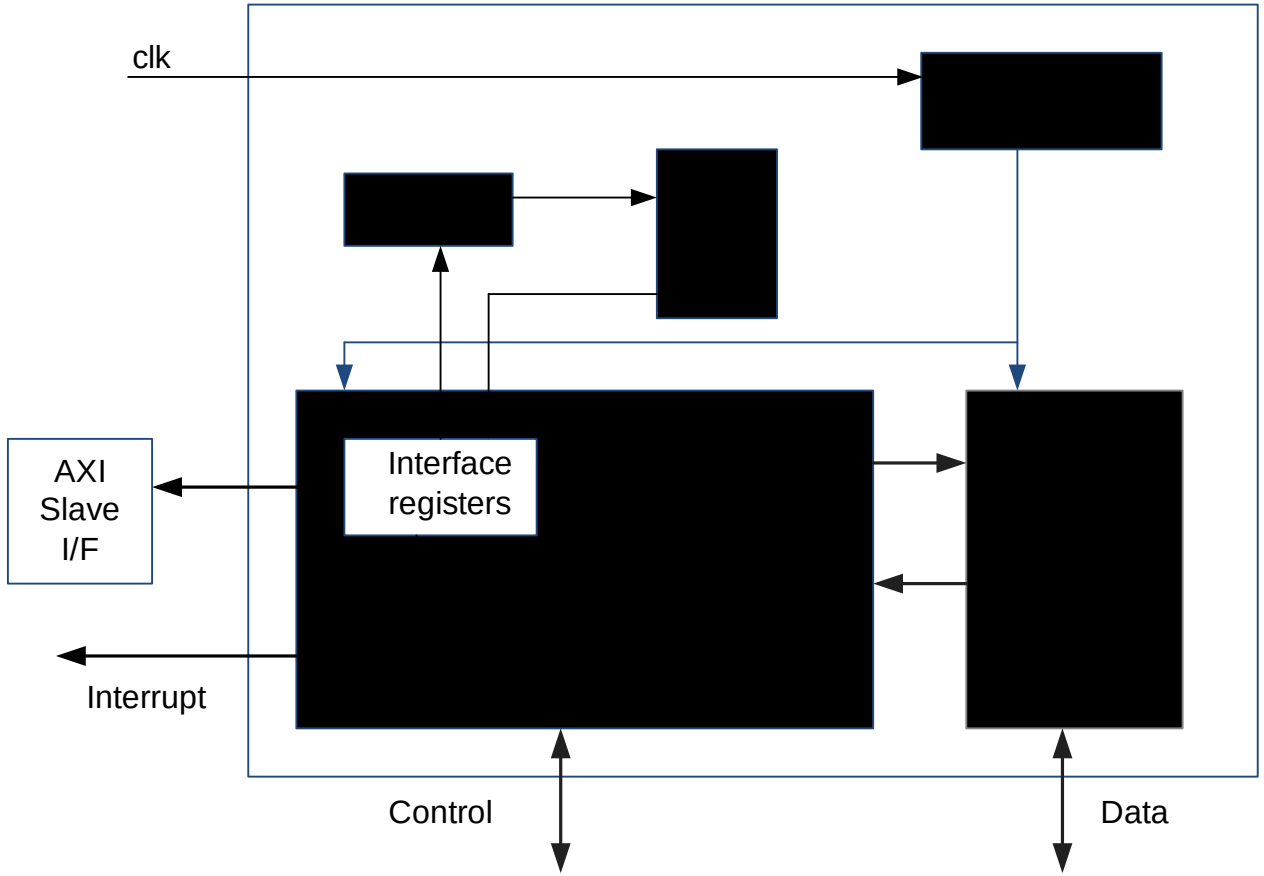
Timer


clk
Clock generator
AXI Slave I/F
Timer
Interrupt
Control
Data

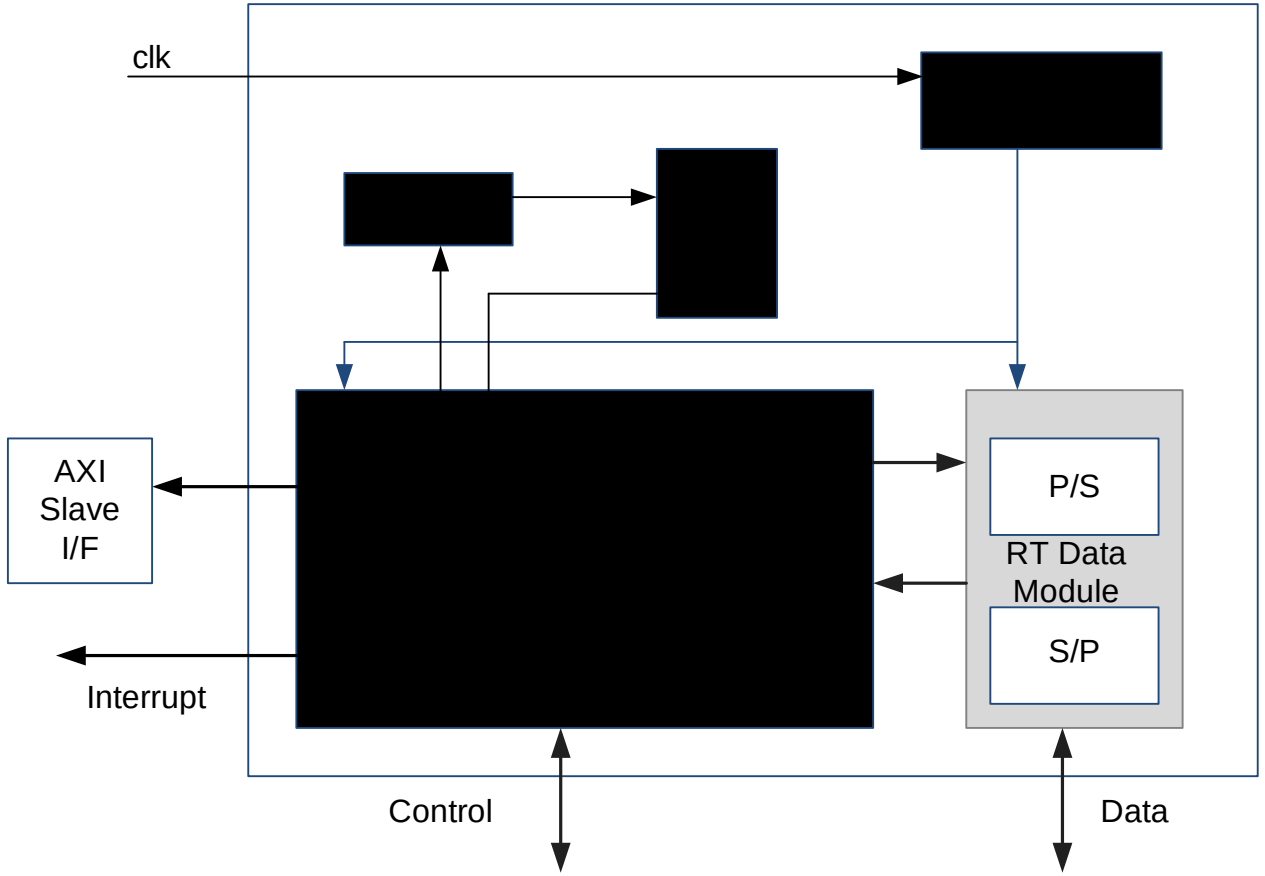
Program Memory



Interface Registers



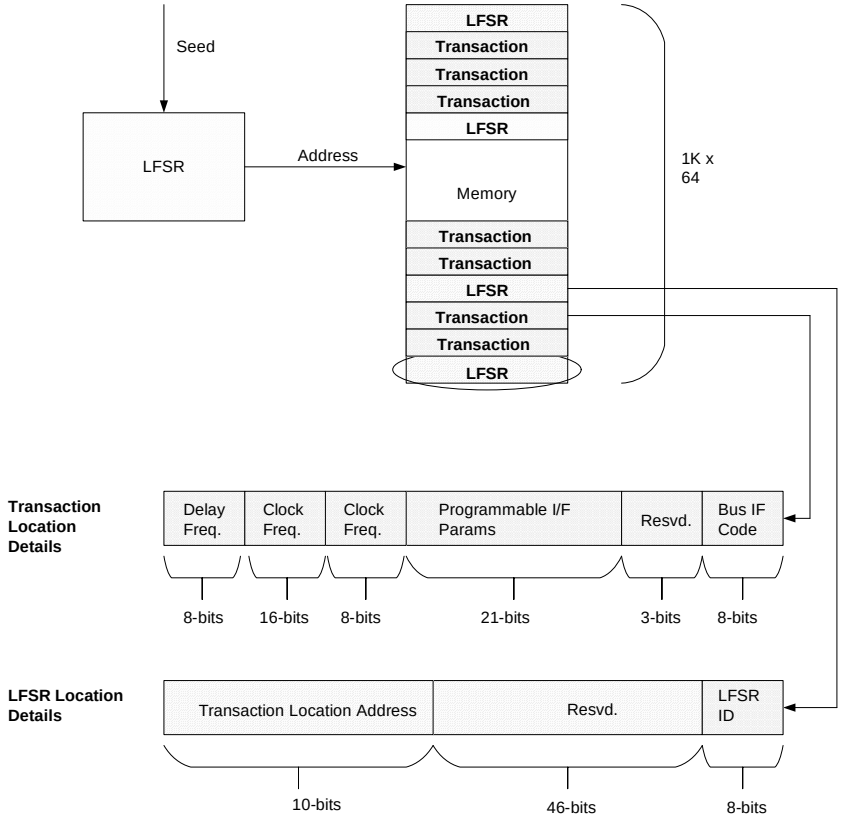
R/T Data Module



Variable Interface Change

- Sequencer allows us to encode different interfaces
- How to hop between the various interfaces
  - To protect IP disclosure through hardware snooping
- A mechanism has been developed
  - Variable Interface Change
- Mechanism allows

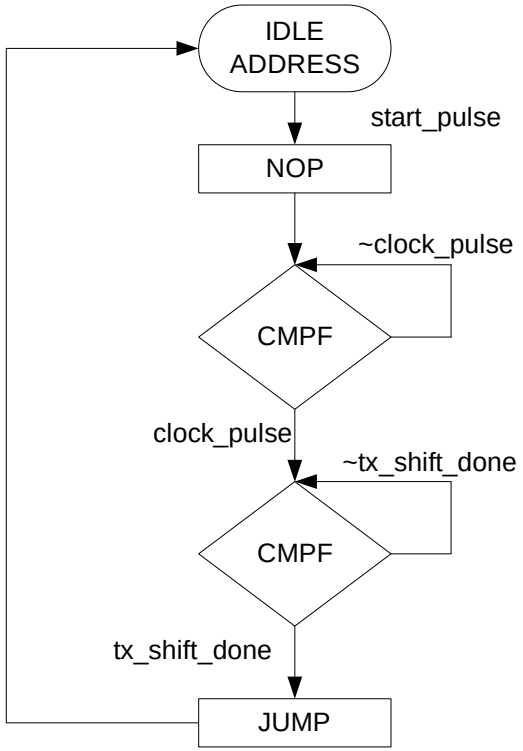
Variable Interface Change (2)



LFSR Transaction Word

Bits	Description
63:56	RI Bus interface code
55:53	Reserved
52:32	Programmable I/F Parameters
31:24	Frame generation frequency. The frequency of the frame generation, e.g. for ASP. The frame generation depends on the word width of the interface.
23:8	RI interface frequency. The frequency at which the RI interface is running. It is some multiple of the clock frequency at which the entire RI module is running.
7:0	Delay between back to back transaction

SPI Write Code Flow



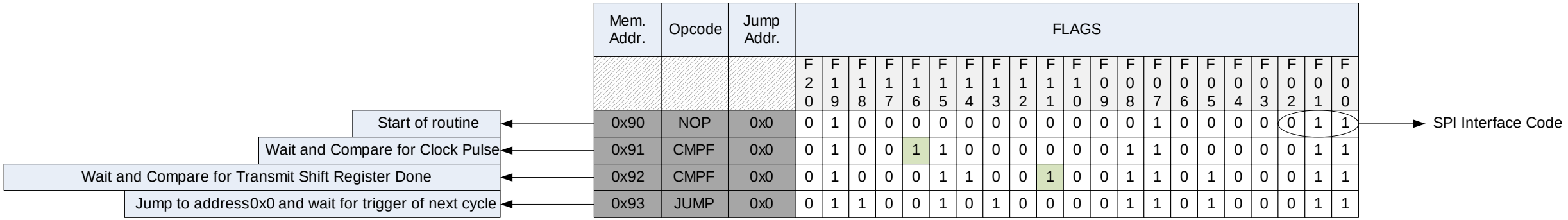
Start Pulse comes from the interface registers. This makes the sequencer address go to the start of the SPI microcode routine.

Wait for Clock Pulse from clock generator in order to align the control and data signals properly with the SPI clock.

Address & Data Phase of SPI

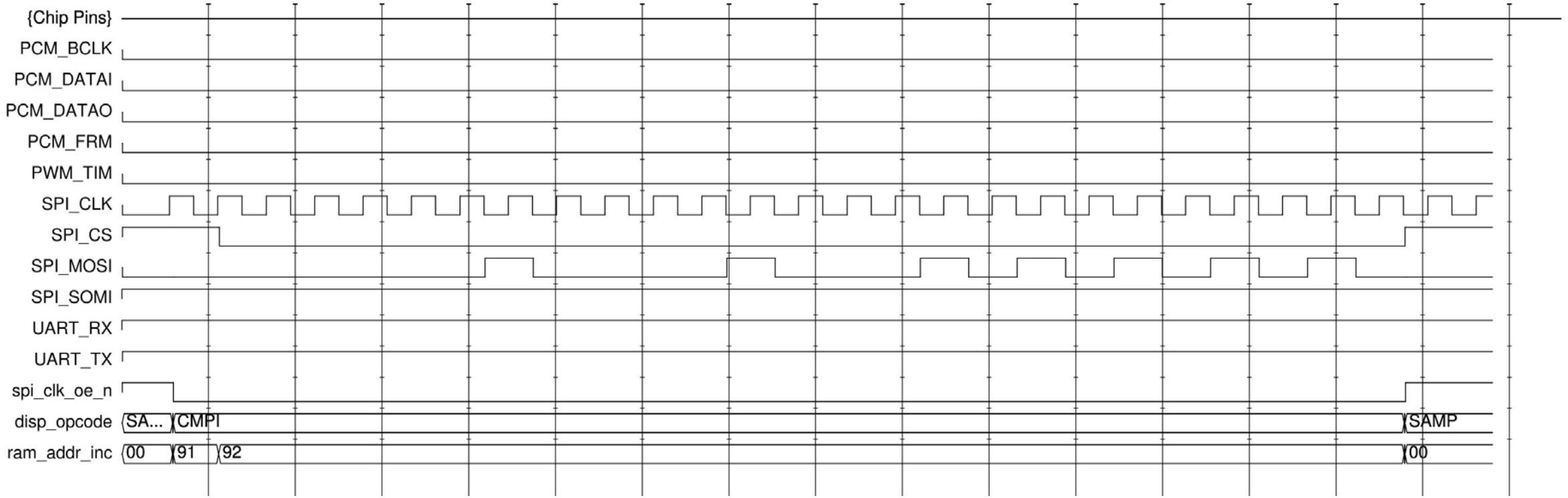
Send out the address location over the SPI data bus, followed by the data. When all the data bits have been shifted out this phase is done. The size of this field is programmable through the interface registers.

SPI Write Micro-Code

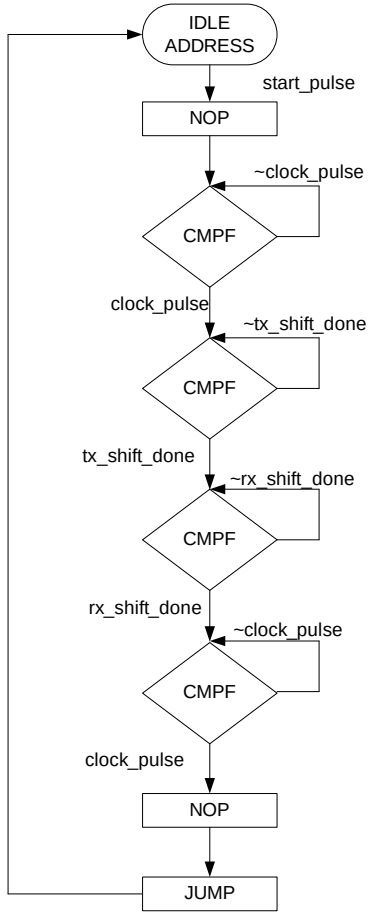


Mem. Addr.	Opcode	Jump Addr.	FLAGS																					
			F20	F19	F18	F17	F16	F15	F14	F13	F12	F11	F10	F09	F08	F07	F06	F05	F04	F03	F02	F01	F00
0x90	NOP	0x0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0x91	CMPF	0x0	0	1	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1
0x92	CMPF	0x0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	1
0x93	JUMP	0x0	0	1	1	0	0	1	0	1	0	0	0	0	1	1	0	1	0	0	0	1	1

SPI Write Cycle



SPI Read Code Flow



Start Pulse comes from the interface registers. This makes the sequencer address go to the start of the SPI microcode routine.

Wait for Clock Pulse from clock generator in order to align the control and data signals properly with the SPI clock.

Address Phase of SPI

Send out the address location over the SPI data bus. When all the data bits have been shifted out this phase is done. The size of this field is programmable through the interface registers.

Read Data Phase of SPI

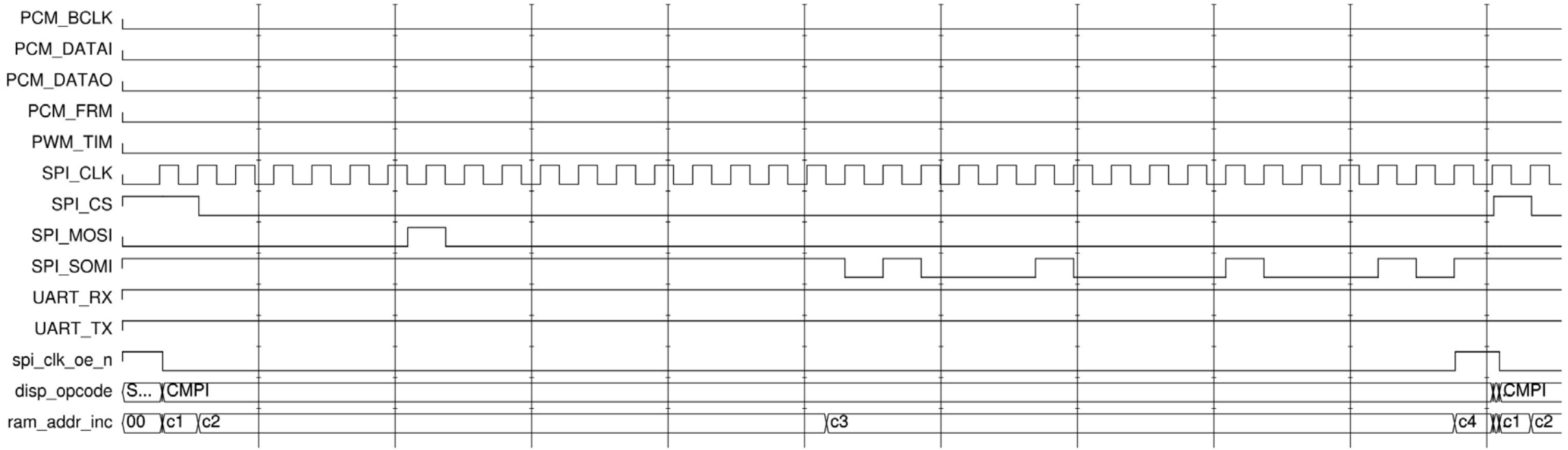
Receive the data from the SPI slave. Once the Receive shift register is full from the requested data this phase is done. The size of this field is programmable through the interface registers.

SPI Read Micro-Code

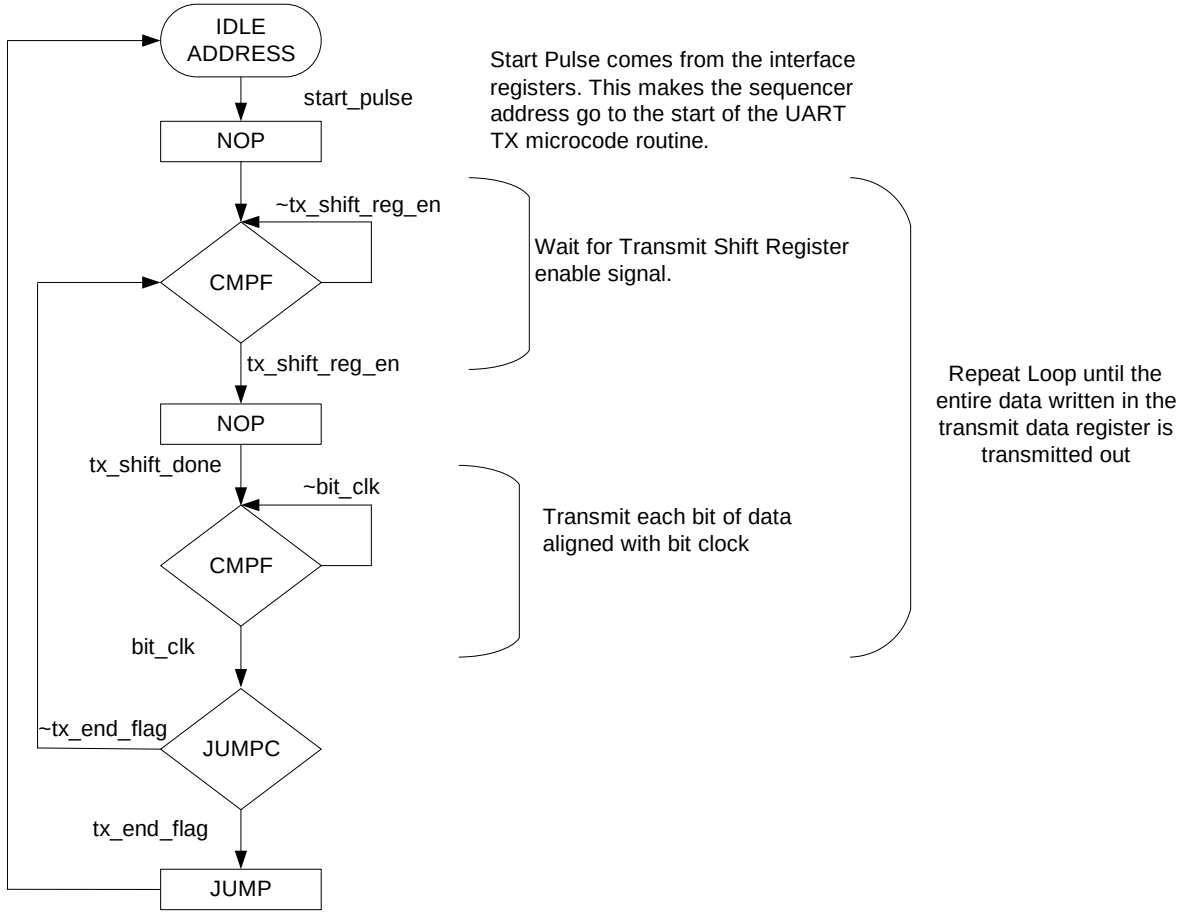
	Mem. Addr.	Opcode	Jump Addr.	FLAGS																					
				F20	F19	F18	F17	F16	F15	F14	F13	F12	F11	F10	F09	F08	F07	F06	F05	F04	F03	F02	F01	F00
Start of routine	0xC0	NOP	0x0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
Wait and Compare for Clock Pulse	0xC1	CMPF	0x0	0	1	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1
Wait and Compare for Transmit Shift Register Done	0xC2	CMPF	0x0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	1
Wait and Compare for Receive Shift Register Done	0xC3	CMPF	0x0	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0	0	0	1	1
Wait and Compare for Clock Pulse	0xC4	CMPF	0x0	0	1	0	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	1	1
Jump to address 0x0 and wait for trigger of next cycle	0xC5	JUMP	0x0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1

SPI Interface Code

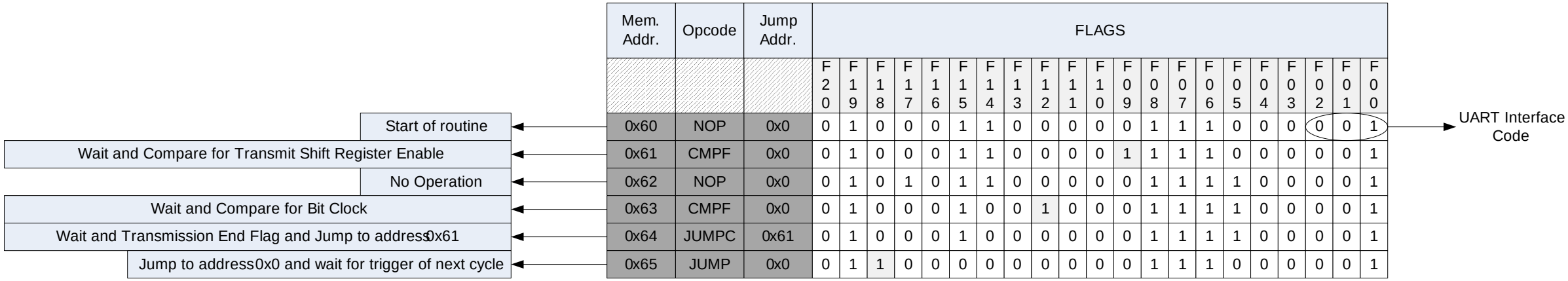
SPI Read Cycle



UART Transmitter Code Flow

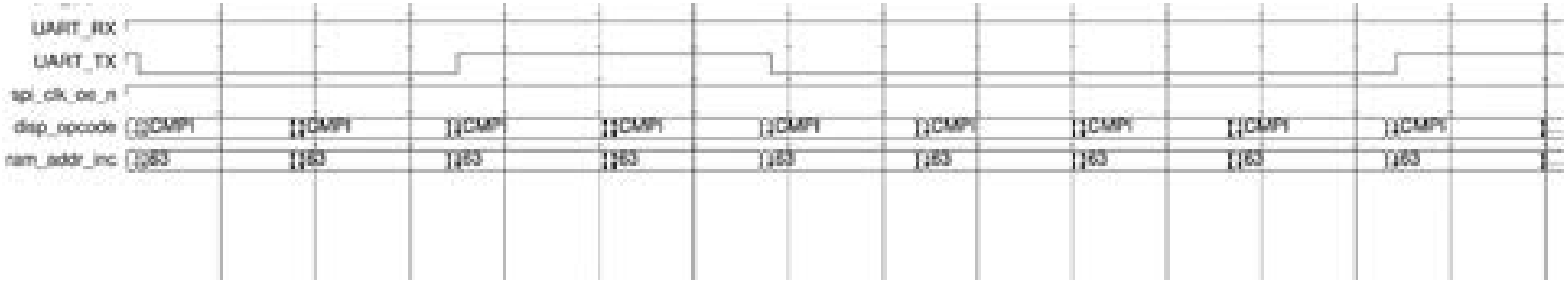


UART Transmitter Micro Code

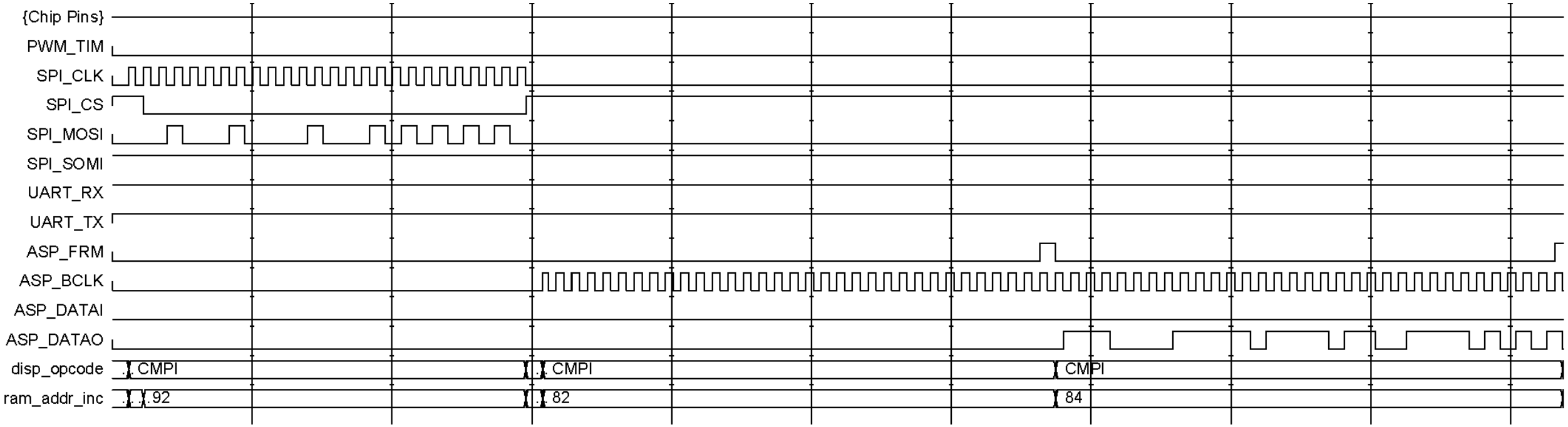


	Mem. Addr.	Opcode	Jump Addr.	FLAGS																					
				F20	F19	F18	F17	F16	F15	F14	F13	F12	F11	F10	F09	F08	F07	F06	F05	F04	F03	F02	F01	F00	
Start of routine	0x60	NOP	0x0	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	UART Interface Code
Wait and Compare for Transmit Shift Register Enable	0x61	CMPF	0x0	0	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	
No Operation	0x62	NOP	0x0	0	1	0	1	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	
Wait and Compare for Bit Clock	0x63	CMPF	0x0	0	1	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	0	0	1	
Wait and Transmission End Flag and Jump to address 0x61	0x64	JUMPC	0x61	0	1	0	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	
Jump to address 0x0 and wait for trigger of next cycle	0x65	JUMP	0x0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	

UART Transmitter



Combination SPI and ASP



Conclusion

- A novel on the fly reconfigurable interface was presented
  - Which can be used to guard against IP leakage through eavesdropping on physical connections in a system or integrated circuit
- A second contribution of our concept is that the reconfiguration is independent of any underlying FPGA technologies

Questions

- Any Questions?