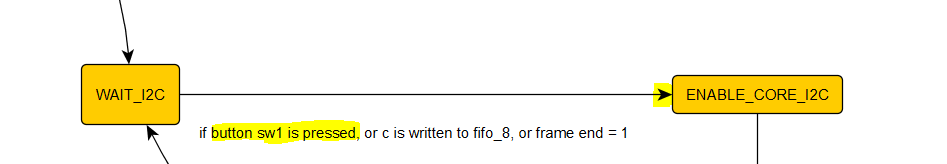
When button sw1 is first pressed:

* changes the i2c state machine to ENABLE\_CORE\_I2C



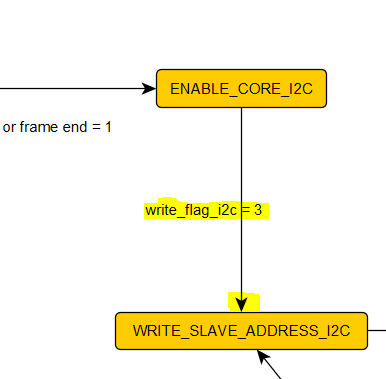
It writes to address CTR the value 0x80 (lines 8 to 13) and proceeds to write the slave address of the camera to the core (line 4 to 7)

1. //sends the enable core command
2. ENABLE\_CORE\_I2C: // step 0
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = WRITE\_SLAVE\_ADDRESS\_I2C;
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = CTR;
12. writedata\_i2c = 8'h80;
13. end
14. end

While being on the ENABLE\_CORE\_I2C state the write flag is incremented (lines 6 to 13)

1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

When write\_flag\_i2c = 3 change state to WRITE\_SLAVE\_ADDRESS\_I2C



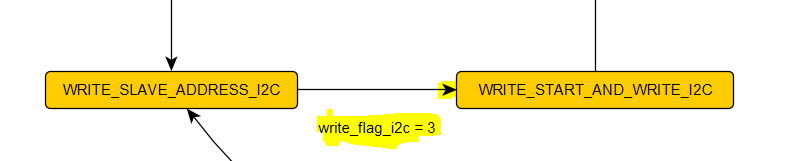
WRITE\_SLAVE\_ADDRESS\_I2C writes to address TXR the value of the camera (lines 8 to 13) and proceeds to send the start and write i2c command to the core (lines 4 to 7)

1. //writes the slave addres to i2c
2. WRITE\_SLAVE\_ADDRESS\_I2C: // step 1
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = WRITE\_START\_AND\_WRITE\_I2C;
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = TXR;
12. writedata\_i2c = address\_slave\_cam\_write;
13. end
14. end

While being on the WRITE\_SLAVE\_ADDRESS state the write flag is incremented (lines 6 to 13)

1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

When write\_flag\_i2c = 3 change state to WRITE\_STAR\_AND\_AND\_WRITE\_I2C



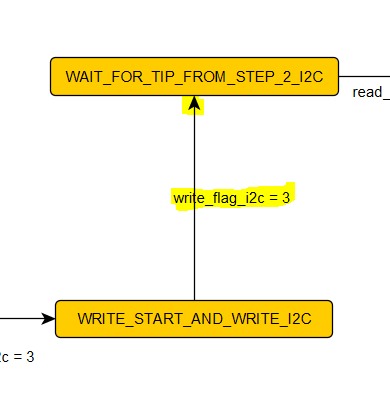
WRITE\_STAR\_AND\_AND\_WRITE\_I2C writes to address CR the value of 0x90 (lines 8 to 13) and proceeds to wait until the i2c command is done (lines 4 to 7)

1. //send the start and write i2c command
2. WRITE\_START\_AND\_WRITE\_I2C: //step 2
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C;
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = CR;
12. writedata\_i2c = 8'h90;
13. end
14. end

While being on the WRITE\_STAR\_AND\_AND\_WRITE\_I2C state the write flag is incremented (lines 6 to 13)

1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

When write\_flag\_i2c = 3 change state to WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C



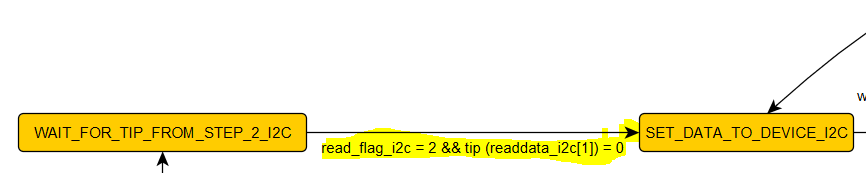
WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C reads register SR (lines 13 to 16), when read it checks bit 1 (line 8) that when is disabled it means the transfer was completed and thus change the state to SET\_DATA\_TO\_DEVICE\_I2C (lines 8 to 11)

1. //wait for the data transmission to be done
2. //then change state
3. WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C: //step 3
4. begin
5. **if**(read\_flag\_i2c == 2)
6. begin
7. address\_i2c = SR;
8. **if**(readdata\_i2c[1] == 0)
9. begin
10. next\_state\_i2c = SET\_DATA\_TO\_DEVICE\_I2C;
11. end
12. end
13. **else**
14. begin
15. address\_i2c = SR;
16. end
17. end

While being on the WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C state the read flag is incremented (lines 4 to 11). Also when being in this state it means we first need to write the upper byte of the address camera register and thus it enables the write upper address flag (lines 18 to 21)

1. //any of these states needs the read flag
2. **if**(current\_state\_i2c == WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C || current\_state\_i2c == WAIT\_TRANSFER\_DONE\_I2C || current\_state\_i2c == WAIT\_FOR\_STOP\_I2C)
3. begin
4. **if**(read\_flag\_i2c == 2)
5. begin
6. read\_flag\_i2c <= 0;
7. end
8. **else**
9. begin
10. read\_flag\_i2c <= read\_flag\_i2c + 1;
11. end
12. end
13. //when data commands its done in this state, upped reg value is needed
14. **if**(current\_state\_i2c == WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C)
15. begin
16. **if**(read\_flag\_i2c == 2)
17. begin
18. **if**(readdata\_i2c[1] == 0)
19. begin
20. write\_upper\_address\_flag <= 1;
21. end
22. end
23. end

When the i2c command is complete (readdata\_i2c[1] = 0) change state to SET\_DATA\_TO\_DEVICE\_I2C



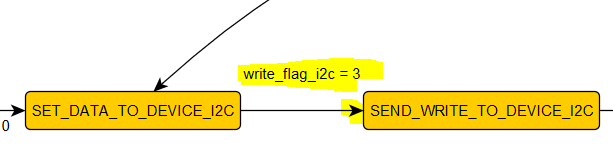
SET\_DATA\_TO\_DEVICE\_I2C writes to address TXR the value of the address of the camera (lines 8 to 21). It evaluates if it needs to write the upper or lower byte of the address of the camera, then it proceeds to send the write i2c command to the core (lines 4 to 7). Register\_index is the current register number to be written, currently 0. Now we are using line 14

1. //sets the corresponding data to be written
2. SET\_DATA\_TO\_DEVICE\_I2C:
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = SEND\_WRITE\_TO\_DEVICE\_I2C; //need flags
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = TXR;
12. **if**(write\_upper\_address\_flag)
13. begin
14. writedata\_i2c = high\_address[register\_index]; // to bevariable
15. end
16. **else** **if**(write\_lower\_address\_flag)
17. begin
18. writedata\_i2c = low\_address[register\_index]; // to bevariable
19. end
21. end
22. end

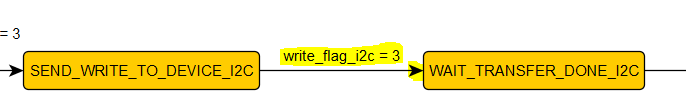
While being on the SET\_DATA\_TO\_DEVICE\_I2C state the write flag is incremented (lines 6 to 13)

1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

When write\_flag\_i2c = 3 change state to SEND\_WRITE\_TO\_DEVICE\_I2C



Afterwards it proceeds to send the command mimicking the previous step. When write\_flag\_i2c = 3 changes state to WAIT\_TRANSFER\_DONE\_I2C



WAIT\_TRANSFER\_DONE\_I2C reads register SR (lines 20 to 23), when read it checks bit 1 (line 8) that when is disabled it means the transfer was completed and thus check which state to change. If the upper byte of the address register was written, change state to SET\_DATA\_TO\_DEVICE\_I2C to write the lower byte. If the lower byte of the address register was writte, change state to SET\_LAST\_DATA\_I2C to write the value to be written to that register. Now we are using line 12

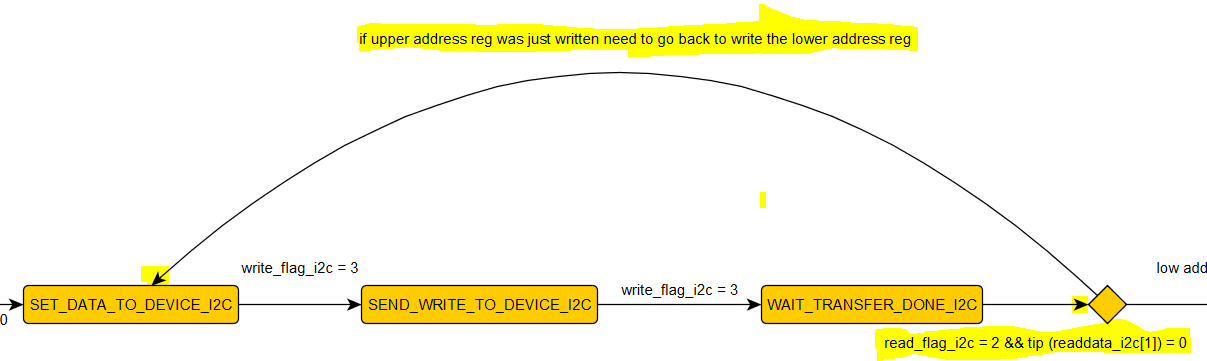
1. //wait for the data transmission to be done
2. //then change state
3. WAIT\_TRANSFER\_DONE\_I2C:
4. begin
5. **if**(read\_flag\_i2c == 2)
6. begin
7. address\_i2c = SR;
8. **if**(readdata\_i2c[1] == 0)
9. begin
10. **if**(write\_upper\_address\_flag)
11. begin
12. next\_state\_i2c = SET\_DATA\_TO\_DEVICE\_I2C; //needs flags
13. end
14. **else** **if**(write\_lower\_address\_flag)
15. begin
16. next\_state\_i2c = SET\_LAST\_DATA\_I2C; //needs flags
17. end
18. end
19. end
20. **else**
21. begin
22. address\_i2c = SR;
23. end
24. end

While being on the WAIT\_TRANSFER\_DONE\_I2C increments the read flag

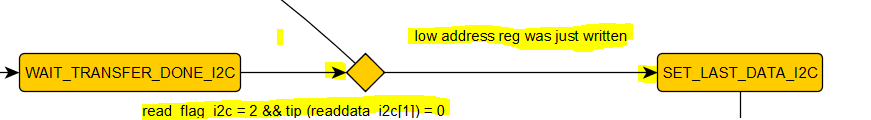
1. //any of these states needs the read flag
2. **if**(current\_state\_i2c == WAIT\_FOR\_TIP\_FROM\_STEP\_2\_I2C || current\_state\_i2c == WAIT\_TRANSFER\_DONE\_I2C || current\_state\_i2c == WAIT\_FOR\_STOP\_I2C)
3. begin
4. **if**(read\_flag\_i2c == 2)
5. begin
6. read\_flag\_i2c <= 0;
7. end
8. **else**
9. begin
10. read\_flag\_i2c <= read\_flag\_i2c + 1;
11. end
12. end

While being on the WAIT\_TRANSFER\_DONE\_I2C, if the transfer is complete, if the upper byte was just written, reset its flag and enable the write lower byte flag (lines to 10 to 14) else if the lower byte was just written reset the variable (lines 16 to 19). Now we are using lines 12-13

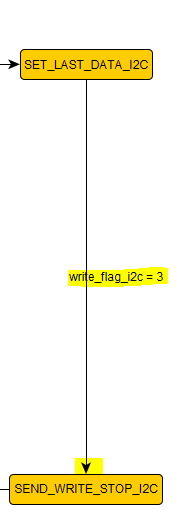
1. //while waiting for the data transfer to be done
2. **else** **if**(current\_state\_i2c == WAIT\_TRANSFER\_DONE\_I2C)
3. begin
4. **if**(read\_flag\_i2c == 2)
5. begin
6. **if**(readdata\_i2c[1] == 0)
7. begin
8. //if the upper address has been written, set the
9. //flag to write the lower reg address
10. **if**(write\_upper\_address\_flag)
11. begin
12. write\_upper\_address\_flag <= 0;
13. write\_lower\_address\_flag <= 1;
14. end
15. //if lower reg address has been written, deassert the flag
16. **else** **if**(write\_lower\_address\_flag)
17. begin
18. write\_lower\_address\_flag <= 0;
19. end
20. end
21. end
22. end

When the i2c command is complete (readdata\_i2c[1] = 0) change state to SET\_DATA\_TO\_DEVICE\_I2C 

Repeat steps from SET\_DATA\_TO\_DEVICE\_I2C except this time in WAIT\_TRANSFER\_DONE\_I2C we are writing the value of the register



SET\_LAST\_DATA\_I2C follows the same procedures as SET\_DATA\_TO\_DEVICE\_I2C. When write\_flag\_i2c = 3 change state to SEND\_WRITE\_STOP\_I2C



SEND\_WRITE\_STOP\_I2C writes to address CR the value 0x50 (lines 8 to 13) and proceeds to wait the command is complete (lines 4 to 7)

1. //send write and stop i2c command
2. SEND\_WRITE\_STOP\_I2C:
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = WAIT\_FOR\_STOP\_I2C;
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = CR;
12. writedata\_i2c = 8'h50;
13. end
14. end

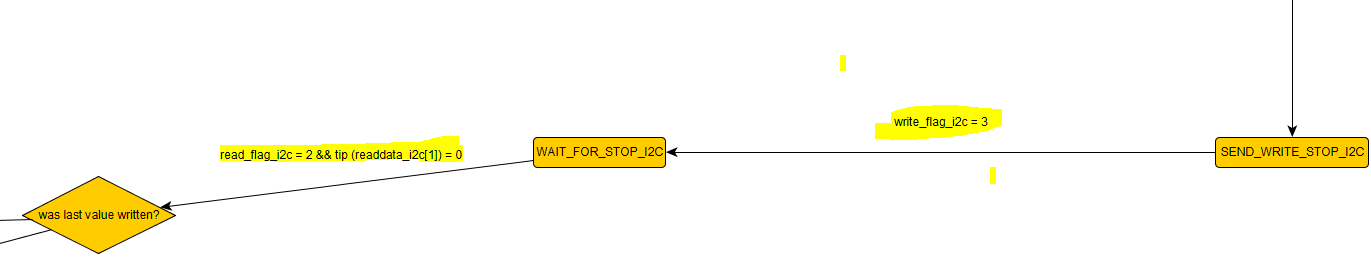
While being on the SEND\_WRITE\_STOP\_I2C state the write flag is incremented (lines 6 to 13)

1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

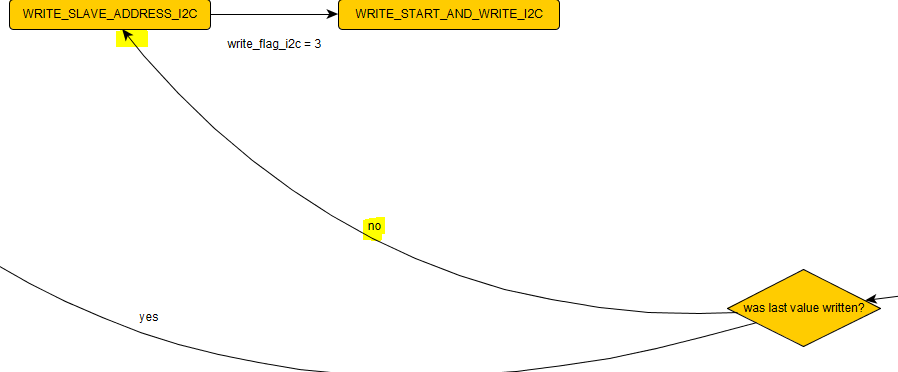
register\_index holds the current register number being written, when finishing sending this command increment the index, if its equal or greater than the total number of registers to be written, set the last transmission flag. In this step we are not setting it.

1. //this states are when the reg value is sent, increments the reg index and check if it should
2. //stop
3. **else** **if** (current\_state\_i2c == SEND\_WRITE\_STOP\_I2C)
4. begin
5. **if**(write\_flag\_i2c == 3)
6. begin
7. register\_index <= register\_index + 1;
8. **if**(register\_index >= (number\_of\_registers -1))
9. begin
10. last\_transmission\_flag <= 1;
11. end //if
12. end //if
13. end //else if

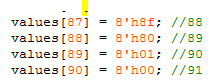
When write flag = 3 change state to WAIT\_FOR\_STOP\_I2C that waits for the stop to be sent, then it checks if the last transmission flag was set.

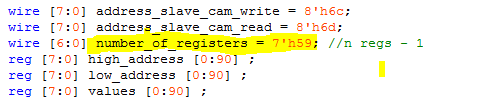


Since it wasn’t we return to WRITE\_SLAVE\_ADDRESS\_I2C to write the register index = 1 of the values



It continues to write the camera register until reaching register\_index = 88, that writes the pattern value of the camera and it stop there

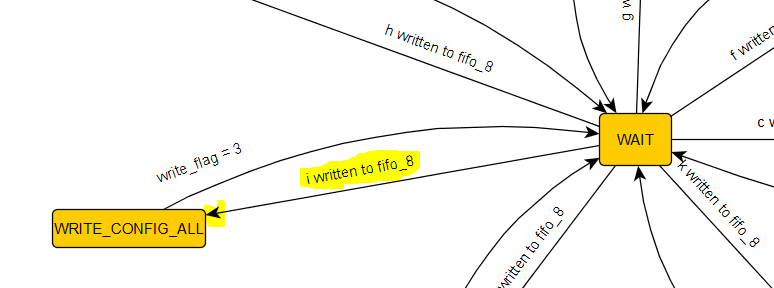




Assuming the button sw1 has been pressed and the galileo has currently running the xilly.c program and its read to receive an input

Sending ‘i’ does the following:

* Changes the state machine to WRITE\_CONFIG\_ALL





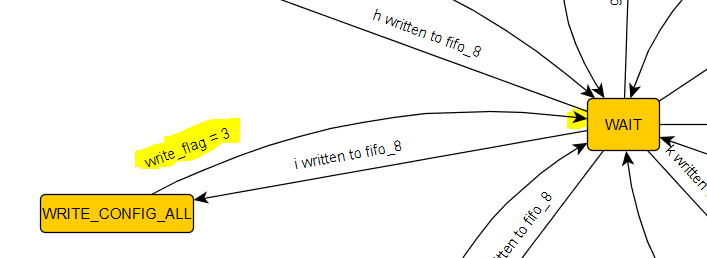
These two snippets work together. When it first enters the state it reads register 1 (lines 15 to 19), when it has been read it enables start write and it writes back the enabled bits 1 and 0 back into register 1 (lines 9 to 14) when finishes writing it goes back to WAIT (lines 5 to 8)

1. //first reads the contents of register 1,
2. //enables bits 1-0, and writes it back
3. WRITE\_CONFIG\_ALL:
4. begin
5. **if**(write\_flag == 3)
6. begin
7. next\_state = WAIT;
8. end
9. **if** (write\_flag <= 3 && start\_write)
10. begin
11. address = 1;
12. write = 1'b1;
13. writedata = from\_readdata;
14. end
15. **if**(read\_flag <= 2 && start\_write == 0)
16. begin
17. read = 1'b1;
18. address = 1;
19. end
20. end

When enters first in WRITE\_CONFIG\_ALL it first increments the read flag (lines 19 to 24). When read flag = 2 (line 6) it resets the read flag (line 8), enables bits 1-0 of the read value (lines 11 to 14), and enables the start write flag (line 17). After, it increments the write flag (lines 31 to 34), when it is equal to 3 it resets the write flag and the start write flag (lines 26 to 30)

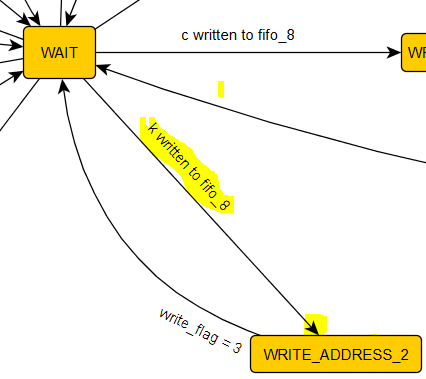
1. //if any of the current states, set the read and write flags to increment
2. **if**(current\_state == WRITE\_ROWS || current\_state == WRITE\_COLUMNS || current\_state == WRITE\_CONFIG\_ALL || current\_state == WRITE\_ENABLE || current\_state == WRITE\_CONFIG
3. || current\_state == WRITE\_ENABLE\_FROM\_FULL || current\_state == WRITE\_ENABLE\_FROM\_EMPTY || current\_state == WRITE\_ENABLE\_FROM\_FE || current\_state == WRITE\_ADDRESS\_2)
4. begin
5. //if the current register value has been read, change to the desired value for writing it back
6. **if**(read\_flag == 2)
7. begin
8. read\_flag <= 0;
9. ...
10. //if its write config all, set the last 2 bits to 1
11. **else** **if**(current\_state == WRITE\_CONFIG\_ALL )
12. begin
13. from\_readdata <= readdata | 32'h0003;
14. end
15. ...
16. //signals that the register has been read and changed and its ready to be written back
17. start\_write <= 1;
18. end
19. **else** **if**(start\_write == 0)
20. begin
21. //increment the read flag and set the delay values
22. read\_flag <= read\_flag + 1;
23. temp\_user\_sw <= (~user\_sw << 8);
24. end
25. //set the write mechanism
26. **if**(write\_flag == 3)
27. begin
28. write\_flag <= 0;
29. start\_write <= 0;
30. end
31. **else** **if**(start\_write)
32. begin
33. write\_flag <= write\_flag + 1;
34. end
35. end

When write flag = 3 it goes back to the wait state



Sending ‘k’ does the following

* Changes the state machine to WRITE\_ADDRESS\_2



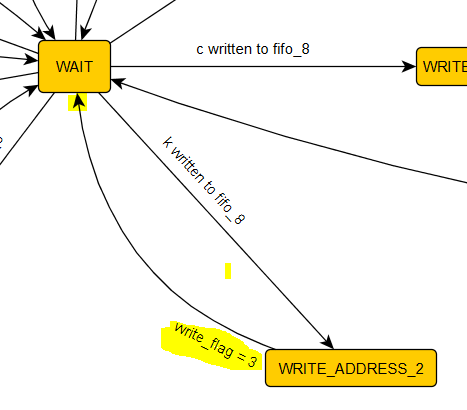
These two snippets work together. When it first enters the state it reads register 2 (lines 16 to 20), when it has been read it enables start write and it writes back the bits 15 to 8 with the values of the dip switch and writes it back into register 2 (lines 10 to 15) when finishes writing it goes back to WAIT (lines 6 to 9)

1. //first reads the contents of register 2,
2. //enables bits 15-8 with the value of the
3. //dip switch, and writes it back
4. WRITE\_ADDRESS\_2:
5. begin
6. **if**(write\_flag == 3)
7. begin
8. next\_state = WAIT;
9. end
10. **if** (write\_flag <= 3 && start\_write)
11. begin
12. address = 2;
13. write = 1'b1;
14. writedata = from\_readdata;
15. end
16. **if**(read\_flag <= 2 && start\_write == 0)
17. begin
18. read = 1'b1;
19. address = 2;
20. end
21. end

When enters first in WRITE\_ADDRESS\_2 it first increments the read flag along with setting up the values of the dip switch with the upper 8 bits of the temp\_user\_sw variable (lines 19 to 24). When read flag = 2 (line 6) it resets the read flag (line 8), writes bits 15-8 of the read value (lines 11 to 14), and enables the start write flag (line 17). After, it increments the write flag (lines 31 to 34), when it is equal to 3 it resets the write flag and the start write flag (lines 26 to 30)

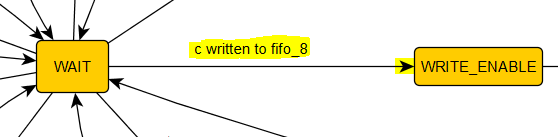
1. //if any of the current states, set the read and write flags to increment
2. **if**(current\_state == WRITE\_ROWS || current\_state == WRITE\_COLUMNS || current\_state == WRITE\_CONFIG\_ALL || current\_state == WRITE\_ENABLE || current\_state == WRITE\_CONFIG
3. || current\_state == WRITE\_ENABLE\_FROM\_FULL || current\_state == WRITE\_ENABLE\_FROM\_EMPTY || current\_state == WRITE\_ENABLE\_FROM\_FE || current\_state == WRITE\_ADDRESS\_2)
4. begin
5. //if the current register value has been read, change to the desired value for writing it back
6. **if**(read\_flag == 2)
7. begin
8. read\_flag <= 0;
9. ...
10. //this writes the values of the dip switch as delay
11. **else** **if**(current\_state == WRITE\_ADDRESS\_2)
12. begin
13. from\_readdata <= readdata | temp\_user\_sw;
14. end
15. ...
16. //signals that the register has been read and changed and its ready to be written back
17. start\_write <= 1;
18. end
19. **else** **if**(start\_write == 0)
20. begin
21. //increment the read flag and set the delay values
22. read\_flag <= read\_flag + 1;
23. temp\_user\_sw <= (~user\_sw << 8);
24. end
25. //set the write mechanism
26. **if**(write\_flag == 3)
27. begin
28. write\_flag <= 0;
29. start\_write <= 0;
30. end
31. **else** **if**(start\_write)
32. begin
33. write\_flag <= write\_flag + 1;
34. end
35. end

When write flag = 3 it goes back to the wait state

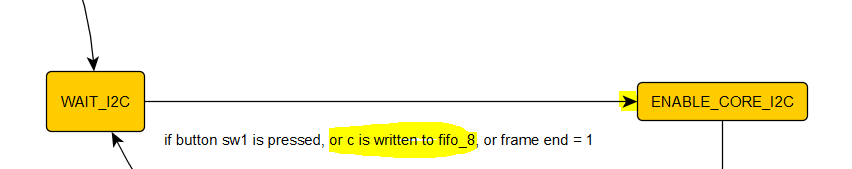


Sending ‘c’ does the following

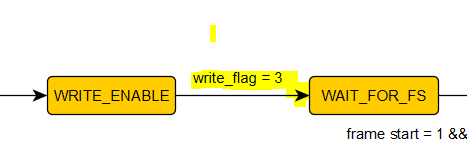
* changes the csi state machine to WRITE\_ENABLE



* changes the i2c state machine to ENABLE\_CORE\_I2C



WRITE\_ENABLE has the same functionality as WRITE\_CONFIG\_ALL and WRITE\_ADDRESS\_2 with the exception that on write\_flag = 3 it goes to WAIT\_FOR\_FS



While the csi machine is changing states, the I2C machine is enabling the core: it writes to address CTR the value 0x80 (lines 8 to 13) and proceeds to write the slave address of the camera to the core (line 4 to 7)

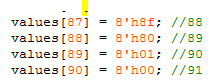
1. //sends the enable core command
2. ENABLE\_CORE\_I2C: // step 0
3. begin
4. **if**(write\_flag\_i2c == 3)
5. begin
6. next\_state\_i2c = WRITE\_SLAVE\_ADDRESS\_I2C;
7. end
8. **else**
9. begin
10. write\_i2c = 1;
11. address\_i2c = CTR;
12. writedata\_i2c = 8'h80;
13. end
14. end

While being on the ENABLE\_CORE\_I2C state the write flag is incremented (lines 6 to 13)

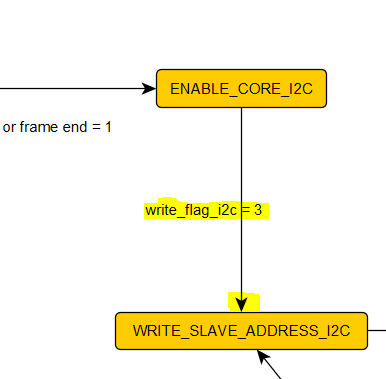
1. //any of these states needs the write flag
2. **if**(current\_state\_i2c == ENABLE\_CORE\_I2C || current\_state\_i2c == WRITE\_SLAVE\_ADDRESS\_I2C || current\_state\_i2c == WRITE\_START\_AND\_WRITE\_I2C
3. ||current\_state\_i2c ==  SET\_DATA\_TO\_DEVICE\_I2C ||current\_state\_i2c == SEND\_WRITE\_TO\_DEVICE\_I2C|| current\_state\_i2c == SET\_LAST\_DATA\_I2C
4. ||current\_state\_i2c == SEND\_WRITE\_STOP\_I2C )
5. begin
6. **if**(write\_flag\_i2c == 3)
7. begin
8. write\_flag\_i2c <= 0;
9. end
10. **else**
11. begin
12. write\_flag\_i2c <= write\_flag\_i2c + 1;
13. end
14. end

It also proceeds to set register index to the 89 (lines 7 to 11) to unsleep the camera directly

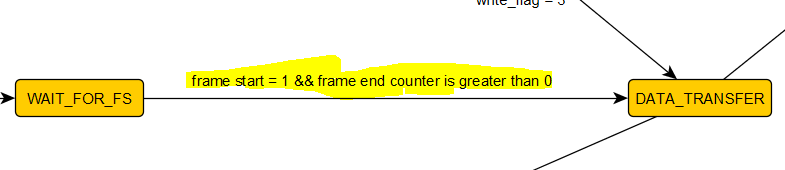
1. //if its waiting, reset the variables, if a write enable command is sent from
2. //fifo 8 (c) set the reg indext to the last value - 1 (unsleep command)
3. **if**(current\_state\_i2c == WAIT\_I2C)
4. begin
5. last\_transmission\_flag <= 0;
6. register\_index <= 0;
7. **if** (user\_r\_read\_8\_data == 8'b01100011) //write enable to csi module, unsleep camera
8. begin
9. register\_index <= 7'h59;
10. last\_transmission\_flag <= 1;
11. end
12. end



When write\_flag\_i2c = 3 change state to WRITE\_SLAVE\_ADDRESS\_I2C



Now the procedures are the same as the start of this document, until the I2C state machine reaches the WAIT\_I2C state. The camera is then enabled and a frame start will produce the CSI state machine to DATA\_TRANSFER



Here data transfer will continue outputting the image to fifo\_32 following the state machine diagram.