RubikBot

3D-Printed Rubik's Cube Solving Robot

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica Y Electrónica
Electrónica Digital II

Elaborado por:

- Óscar Julian Umaña B. (ojumanab@unal.edu.co).
- Juan Diego Ocampo M. (jdocampom@unal.edu.co).
- Raúl Felipe Morales R. (rafmoralesri@unal.edu.co).

1. Manual del Usuario

El RubikBot está diseñando e implementado en el procesador Lattice Mico 32, que se ejecuta en una tarjeta de desarrollo Nexys4DDR, la cual cuenta con una FPGA Artix 7 de Xilinx.

Está conformado por una estructura impresa en 3D con 4 brazos, de los cuales cada uno tiene dos servomotores incorporados en él, uno de ellos se encarga de girar la “mano” en el extremo del brazo (que es utilizada para sujetar el cubo) hacia los lados, mientras que el otro tiene como función mover el brazo hacia atrás o hacia adelante, según sea necesario.

Para determinar el estado inicial de cada una de las seis caras del cubo se tiene una cámara ubicada unos 10 cm sobre el cubo hacia el centro, la cual se encarga de tomar fotografías de cada una de las caras y enviarlas al computador, el cual, mediante un código en Python realizará el procesamiento de imágen necesario para reconocer el color y posición de cada pieza del cubo en cada cara para así determinar su estado inicial.

Posteriormente, el programa determina un algoritmo para solucionar el cubo y mediante un comando en Terminal se envía la orden de resolver el cubo.

1.1 Requisitos del sistema

- Tener instalado el software *Xilinx ISE DS 14.7.*
- Instalar los Toolchains del procesador LM32. Para más información ir al siguiente enlace: *https://github.com/Fabeltranm/lm32_soc_HDL/wiki/Instalaci%C3%B3n--y-configuraci%C3%B3n-del-Toolchain-para-el-procesador-LM32*
- Instalar iVerilog y GTKWave.

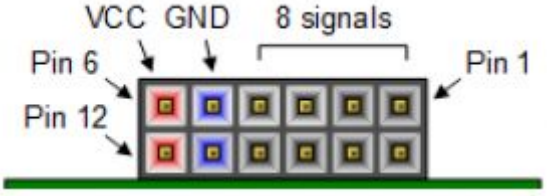
- Tener instalado Python 2.7 o superior.
- Instalar los siguientes paquetes y librerías de Python: CV2, PyCuber, PIL, PySerial, PIP, Numpy, Struct, Time, SYS, OS y IPython.
- Tener instalados los drivers Adept y Runtime de Digilent para la tarjeta Nexys4DDR.
- Descargar el código fuente del programa desde el siguiente enlace: *https://github.com/ltherreraro/CubeRubik/tree/master/HW/04GRUPO4/Source%20Code*

1.2 Procedimiento

En primer lugar, se debe conectar cada uno de los servos a la PCB en su respectivo espacio (desde el SERVO_0 hasta el SERVO_7).

Conectar los cables de la señal de los servos desde la PCB a la tarjeta Nexys4DDR de la siguiente forma:

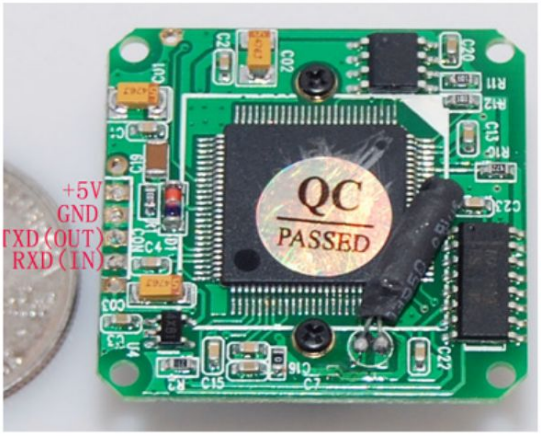
```
PCB RubikBot v2.0 SERVOS -----> Nexys4DDR PMOD JXADC

SERVOS S0 -----> Pmod JXADC_1
SERVOS S1 -----> Pmod JXADC_7
SERVOS S2 -----> Pmod JXADC_2
SERVOS S3 -----> Pmod JXADC_8
SERVOS S4 -----> Pmod JXADC_3
SERVOS S5 -----> Pmod JXADC_9
SERVOS S6 -----> Pmod JXADC_4
SERVOS S7 -----> Pmod JXADC_10
GND -----> Pmod JXADC_5 o Pmod JXADC_7
```



Ahora, conectar la cámara correctamente:

```
VCC -----> VCC (CAMERA_IN PCB)
GND -----> GND (CAMERA_IN PCB)
TX  -----> TX (CAMERA_IN PCB)
RX -----> RX (CAMERA_IN PCB)
```

```
TX (CAMERA_OUT PCB) -----> Pmod JD_1
RX (CAMERA_OUT PCB) -----> Pmod JD_2
```



Ahora, es indispensable programar la FPGA con el procesador LM32 que se ha descargado. Para ello, se abrirá una Terminal en la carpeta del proyecto RubikBot mediante el siguiente comando:

```
cd ~/<DestinationFolder>/RubikBot/lm32-rubikbot-master/
```

Reemplazar <DestinationFolder> por la carpeta en la que está guardado el código del proyecto. Ej: `cd ~/Downloads/RubikBot/lm32-rubikbot-master/.` Finalmente:

```
djtgcfg enum
djtgcfg init -d Nexys4DDR
djtgcfg prog -d Nexys4DDR -i 0 -f system.bit
```

Ahora, en otra Terminal se debe acceder a la carpeta del código en Python (UJLM).

```
cd ~/<DestinationFolder>/RubikBot/UJLM-master/
```

Para ejecutar el código se escribe el siguiente comando:-

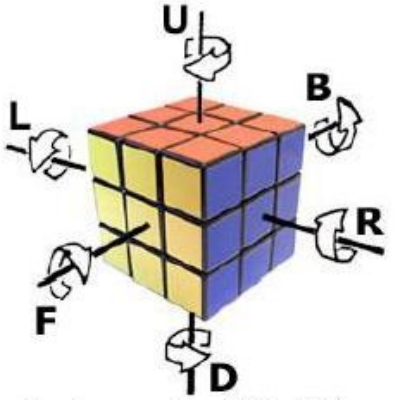
```
python example.py
```

En la Terminal aparecerá el mensaje "`Esperando Bot...`" Es necesario presionar el botón CPU_RESET de la tarjeta Nexys4DDR hasta que aparezca "`Initialising...`"

Posteriormente se selecciona alguna de las siguientes funciones del código en Python, según lo que se desee hacer.

- `init:` Lleva los brazos hacia adelante para agarrar el cubo.

- `home:` Lleva los brazos hacia atrás.
- `mover:` Como se indica en la figura dependiendo la cara que se quiera mover se escribe en la consola la dirección del movimiento. Ejemplo: mover Ra, mueve la cara “R” hacia la derecha y mover R’ mueve la cara “R” hacia la izquierda.



- `calibra:` Permite realizar ajustes en el ciclo útil de los servos, lo cual permite realizar una calibración en caso de ser necesario la sintaxis es: `calibra dir (derecha, izquierda…) del brazo #(1,2,3,4) a valor(3 a 46).`
- `ver cubo:` Toma la foto de cada cara del cubo para determinar el estado inicial del cubo.
- `crear algoritmo:` Genera el algoritmo para solucionar el cubo.
- `resolver cubo:` Ejecuta el algoritmo generado.

2. Especificaciones Técnicas

2.1) Módulo de Servomotores

- Referencia: Micro Servomotor SG90
- Protocolo de comunicación: PWM
- Datasheet: http://www.mactronica.com.co/servomotor-micro-servo-sg90-trower-pro-12882802xJM
- Precio (unidad): COP 7,000
- Enlace de compra: http://www.mactronica.com.co/servomotor-micro-servo-sg90-trower-pro-12882802xJM

2.1.1) PWM

La técnica Pulse Width Modulation consiste en variar el ancho de pulso de una señal cuadrada de tensión con el objetivo de controlar la cantidad de potencia administrada a los componentes o elementos electrónicos conectados.

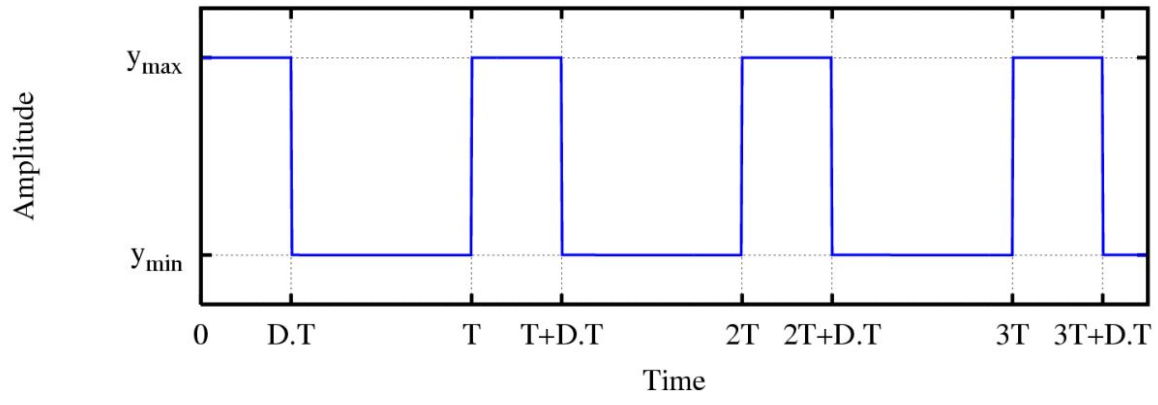


Diagrama de Caja Negra

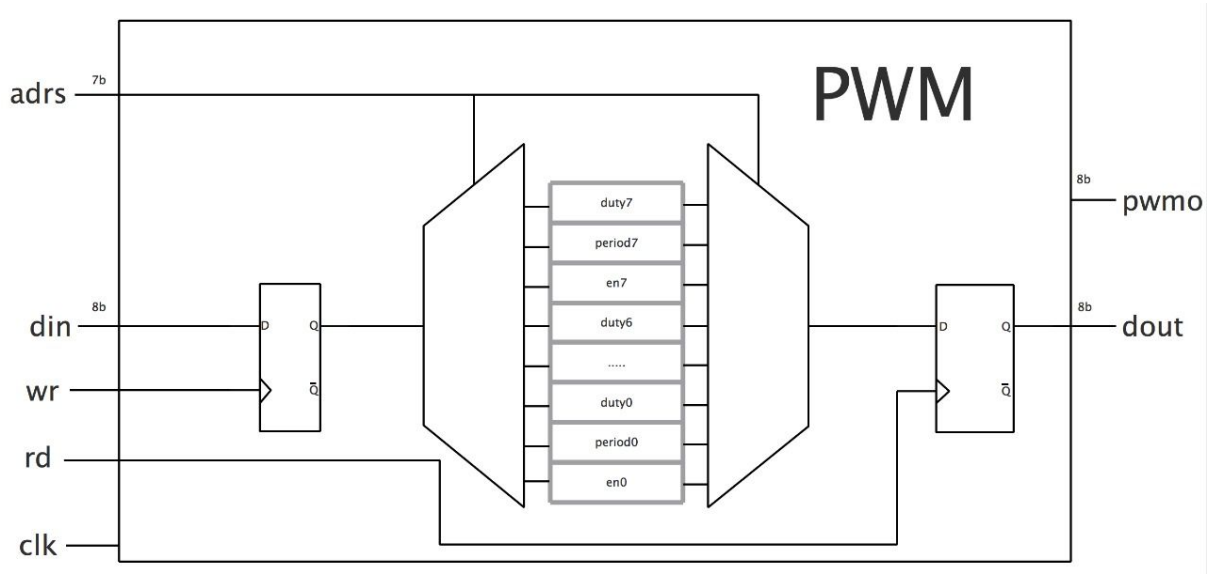
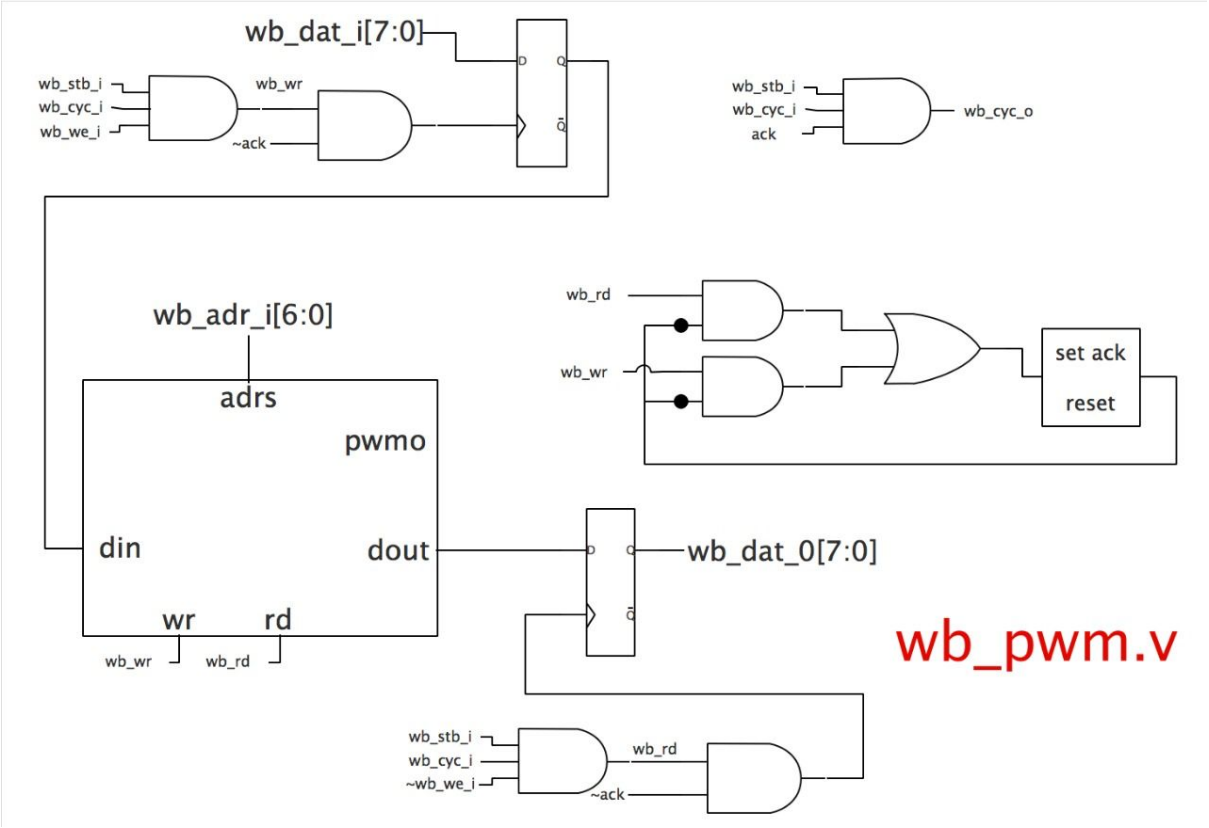


Diagrama (Wishbone):



Mapa de Memoria

READ	WRITE	ADDRESS
DutyCycle_7	DutyCycle_7	00 5C
Period_7	Period_7	00 58
	Enable_7	00 54
DutyCycle_6	DutyCycle_6	00 50
Period_6	Period_6	00 4C
	Enable_6	00 48
DutyCycle_5	DutyCycle_5	00 44
Period_5	Period_5	00 40
	Enable_5	00 3C
DutyCycle_4	DutyCycle_4	00 38
Period_4	Period_4	00 34
	Enable_4	00 30
DutyCycle_3	DutyCycle_3	00 2C
Period_3	Period_3	00 28
	Enable_3	00 24
DutyCycle_2	DutyCycle_2	00 20
Period_2	Period_2	00 1C
	Enable_2	00 18
DutyCycle_1	DutyCycle_1	00 14
Period_1	Period_1	00 10
	Enable_1	00 0C
DutyCycle_0	DutyCycle_0	00 08
Period_0	Period_0	00 04
	Enable_0	00 00

Tabla 1. Mapa de memoria para el periférico PWM.

Los servos SG90 trabajan con un periodo de la señal de 20 ms, y la posición del brazo se indica con su ciclo útil. Se usaron 8 servos, para los cuales se puede

asignar el valor de su ciclo útil, así como su periodo en caso de querer trabajar con un servo que maneje un periodo diferente de trabajo. Se escoge el servo que se quiere trabajar en valores que van del 0 al 7.

2.1.2) Brazos

Se implementó unos brazos los cuales se basan en el uso de dos servos para mover el brazo hacia adelante, atrás, y rotar una cara del cubo hacia la izquierda o derecha, para que llevara a cabo una instrucción sin que se colisionara con otro brazo se emplea un tiempo de 1 ms entre instrucción excepto en el caso de requerir cambiar la cara del cubo en dicho caso no había tiempo de retardo para que se ejecutarán las instrucciones al tiempo. Se implementan cuatro brazos.

2.2) Módulo de Cámara

- Referencia: LinkSprite LS-Y201 Serial Port Camera Module
- Protocolo de comunicación: UART
- Datasheet: https://www.sparkfun.com/datasheets/Sensors/Imaging/1274419957.pdf
- Precio: COP 120,000
- Enlace de compra: https://www.amazon.com/gp/product/B016PZIP3C

Diagrama de Caja Negra

Cámara LS-Y201:

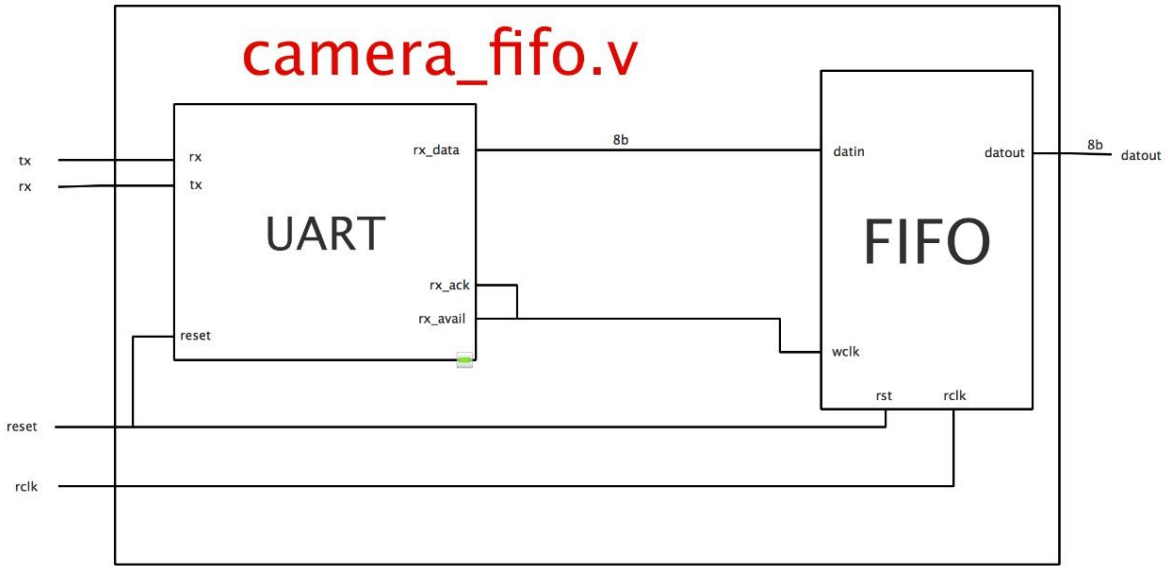
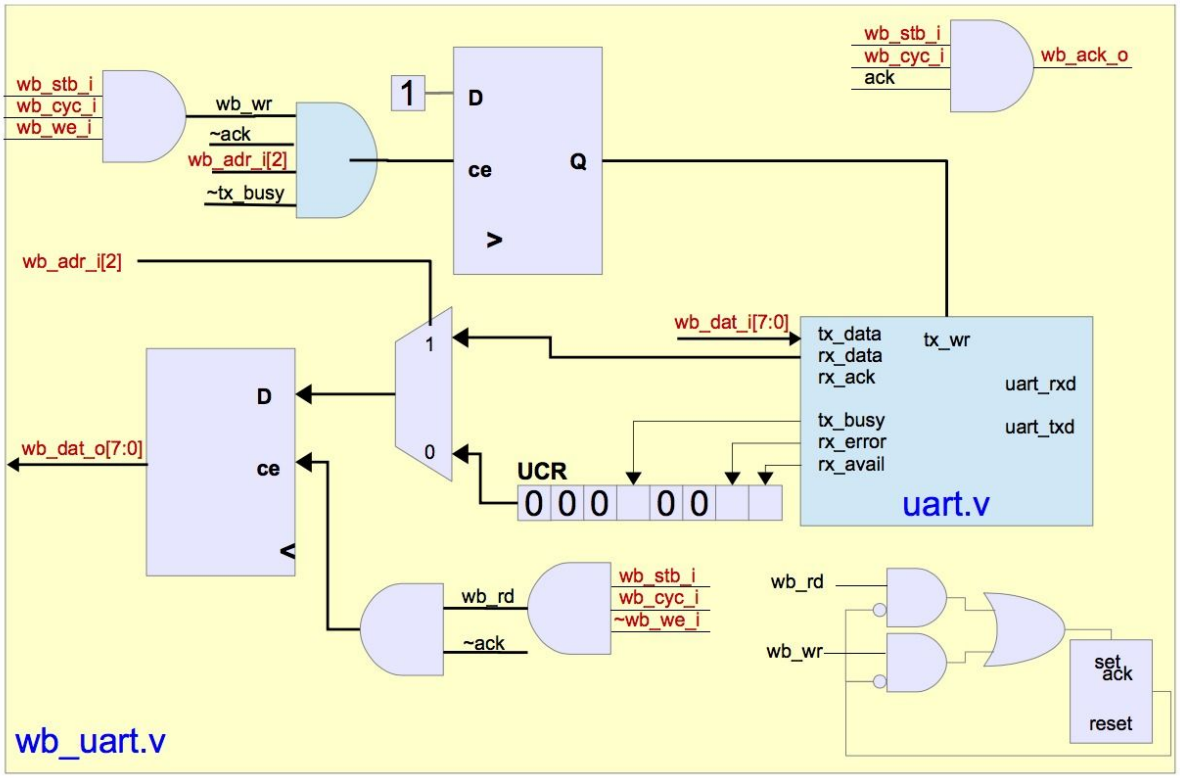


Diagrama (Wishbone):



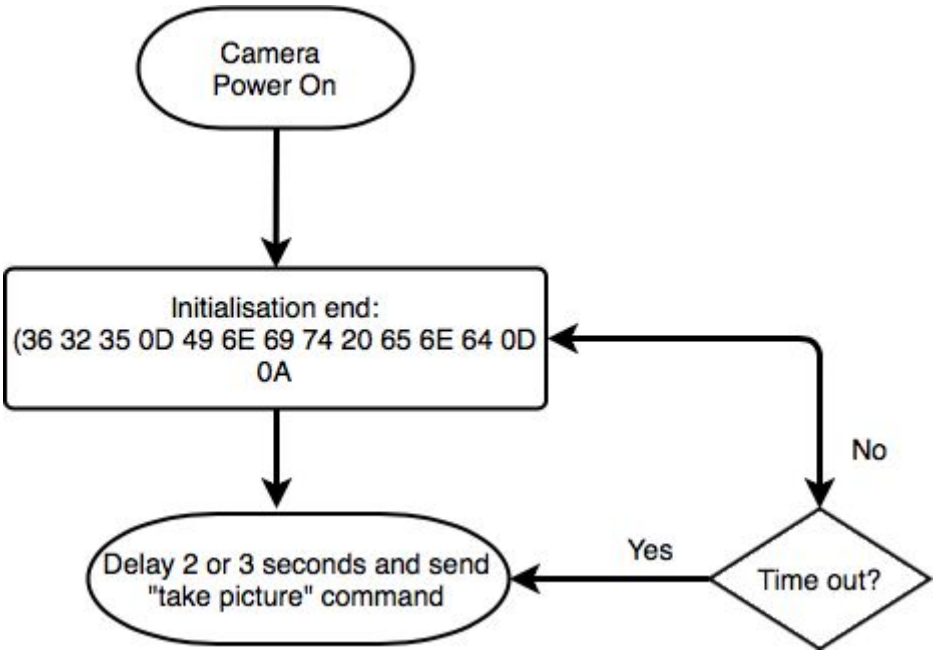
Mapa de Memoria (FIFO):

READ	WRITE	ADDRESS
	Reset	00 0C
	DataIn	00 08
DataOut		00 04
Data		00 00

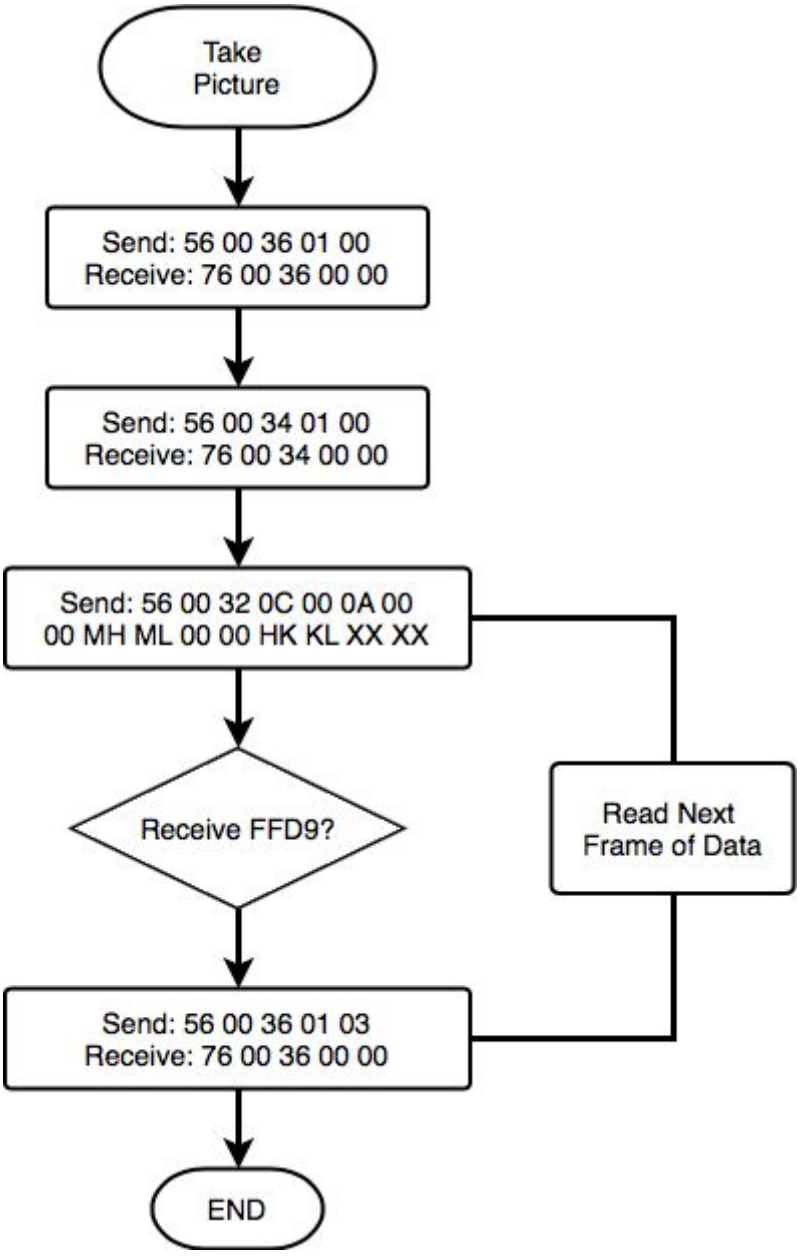
Tabla 3. Mapa de memoria para el periférico FIFO.

Descripción Funcional

Cámara LS-Y201:



Capture a JPEG Picture:



Communication Protocol:

Reset	
Command (HEX)	**Return (HEX)**
56 00 26 00	76 00 26 00
Take Picture	
Command (HEX)	**Return (HEX)**
56 00 36 01 00	76 00 36 00 00
Read JPEG File Size	
Command (HEX)	**Return (HEX)**
56 00 32 0C 00 0A 00 00 MH ML 00 00 KH KL XX XX	76 00 32 00 00 (Interval Time) FF D8, …, 76 00 32 00 00
Interval Time	
00 00 MH ML –> Starting Address	
00 00 KH KL –> Length of JPEG File	
MSB First and LSB Last	
Stop Taking Pictures	
Command (HEX)	**Return (HEX)**
56 00 36 01 03	76 00 36 00 00
Compression Ratio	
Command (HEX)	**Return (HEX)**
56 00 31 05 01 01 12 04 XX	76 00 31 00 00 XX (XX normally is 36)
XX: 0X00 to 0XFF	
Image Size	
Command (HEX)	**Return (HEX)**
56 00 31 05 04 01 00 19 11 (320*240)	76 00 31 00 00
56 00 31 05 04 01 00 19 00 (640*480)	76 00 31 00 00
56 00 31 05 04 01 00 19 22 (160*120)	76 00 31 00 00
Power Saving	
Entering Power Saving Command (HEX)	**Return (HEX)**
56 00 3E 03 00 01 01	76 00 3E 00 00
Entering Power Saving Command (HEX)	**Return (HEX)**
56 00 3E 03 00 01 00	76 00 3E 00 00
Change BAUD Rate	
Command (HEX)	**Return (HEX)**
56 00 24 03 01 XX XX	76 00 24 00 00
XX XX	Data Rate
AE C8	9600
56 E4	19200
2A F2	38400
1C 4C	57600
0D A6	115200

Mapa de Memoria

READ	WRITE	ADDRESS
RxTx	RxTx	00 04
UCR		00 00

Se emplea por defecto la cámara con un BAUD rate de 38400, envía la foto con un tamaño de 320X240 pixels. Se utilizan las siguientes funciones para controlar la cámara, las cuales envían por UART los comandos y luego leen y comparan el retorno de la cámara para confirmar que se envió correctamente.

- *reset():* Resetea la cámara, la cual queda lista para tomar una nueva foto.
- *takepicture():* Toma la foto.
- *getsize():* Recibe la longitud del archivo JPEG.
- *sendpicture():* Envía la foto por medio del UART y se procesa en un computador, por medio de Python.
- *stoptaking():* Después de recibir el archivo JPEG envía el comando para dejar de tomar la foto.

2.2.1) Mapeo

El mapeo del cubo se hace por medio de la clase *face*, para el algoritmo que se se usa es necesario definir las caras del cubo, se empieza por la cara que tiene el cuadrado central blanco hacia el frente y la cara que tiene el cuadrado rojo hacia arriba, se definen las siguientes funciones:

- *Face():* Permite mover el cubo hacia una cara determinada por el usuario.
- *facepictureinit():* Ubica los brazos de manera que se despeje la cara de arriba del cubo para tomar la foto sin que las manos afecten los datos.
- *facepictureend():* Ubica los brazos a la posición de *init().*
- *R1()-L1()-R11()-L11():* Ejecutan los movimientos para cambiar la cara del cubo.

3. Hardware:

La implementación del pwm en verilog se hizo con dos módulos: counter.v y pwm.v

El módulo counter.v es el que se encarga de la generación de un pwm. Este se instancia 8 veces en pwm.v para así tener 8 señales independientes.
También se hizo un módulo FIFO.v el cual se encargaría de enviar el archivo JPEG para su procesamiento, pero se decidió no implementarlo ya que el tamaño de la FIFO no era suficiente, y cuando se implementó con el tamaño de 8 bits de longitud, en una pila de 128, corrompía el archivo JPEG.

counter.v

Este módulo tiene las siguientes entradas y salidas:

```
module counter(clk,period,duty,state,en);

    input clk;
    input en;
    input [7:0] period;
    input [7:0] duty;
    output reg state;
```

Donde:

- `clk` es la entrada de del reloj que se esté usando.
- `en` es la que habilita el funcionamiento de la salida de la salida state.
- `period` es el periodo al que se desea trabajar el pwm.
- `duty` es el ciclo útil al que se desea trabajar el pwm.
- `state` es la salida del pwm, las cual solo cambia al estar habilitada la entrada `en`.

En este módulo lo que se hace es contar los ciclos de reloj que son necesarios para 1us. Para luego implementarlo en la lógica del mismo. Teniendo los valores de periodo y ciclo útil, lo que se hace es poner en alto la salida state durante el tiempo del ciclo útil y mantenerlo abajo en lo que sería el periodo menos el ciclo útil.

pwm.v

Este módulo cumple la función de instanciar 8 modulos counter y de actuar como multiplexor para los valores de cada uno de estos módulos. Con este se puede ingresar los valores para las características de cada uno de los PWM, además de

leer los valores actuales de las mismas. Este módulo tiene las siguientes entradas y salidas:

```
module pwm(clk,rst,rd,wr,din,adrs,dout,pwmo);

      input clk;
      input rst;
      input rd;
      input wr;
      input [7:0] din;
      input [6:0] adrs;
      output reg [7:0] dout;
      output [7:0] pwmo;
```

Donde:

- `clk` es la entrada de reloj.
- `rst` es la entrada para poner en cero todos los valores de los pwm.
- `rd` es la entrada con la que indicamos si queremos leer los valores de las señales.
- `wr` es la entrada con la que indicamos si queremos escribir los valores de las señales.
- `din` es el valor que queremos ingresar a la característica seleccionada con `adrs`.
- `adrs` es la entrada con la que se indica la característica y el PWM a modificar o leer según wr o rd. La siguiente tabla indica los valores para cada característica.
- dout es la salida por la cual obtenemos el valor que queremos leer al habilitar `wr`.
- `pwmo` es la salida de los PWM siendo pwm[0] la primera señal y pwm[7] la última..

Característica	adrs(hex)
duty7	00 5C
period7	00 58
en7	00 54
duty6	00 50
period6	00 4C
en6	00 48
duty5	00 44
period5	00 40
en5	00 3C
duty4	00 38
period4	00 34
en4	00 30
duty3	00 2C
period3	00 28
en3	00 24
duty2	00 20
period2	00 1C
en2	00 18
duty1	00 14
period1	00 10
en1	00 0C
duty0	00 08
period0	00 04
en0	00 00

En este se sigue la lógica de un multiplexor. Según las entradas wr y rd se lee o se escribe la característica que se desea. Además para poder leer y escribir dichos valores se almacenan en registros todo lo ingresado, para así leerlos cuando se necesite.

FIFO.v

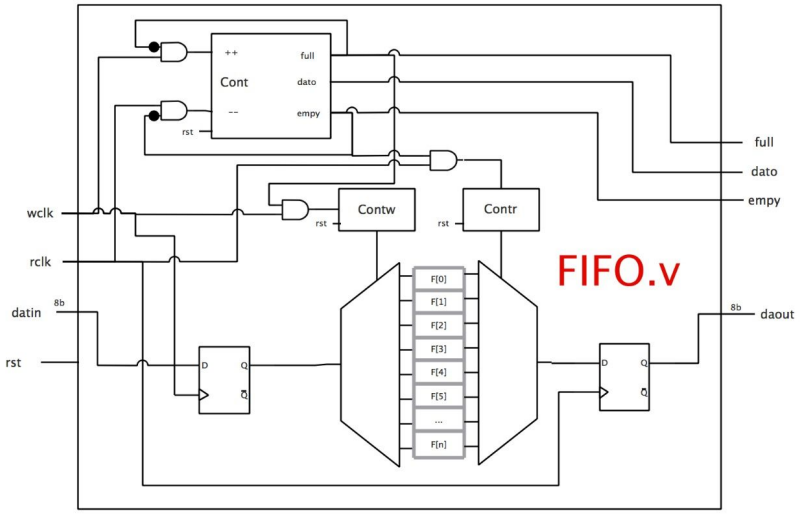
Este módulo cumple la función de tener una pila de datos en el cual el primer dato en escribirse es el primer dato en leerse, la longitud y el tamaño se pueden definir como parámetro, lo que nosotros utilizamos para el archivo JPEG fué de longitud de 8 bits y un tamaño de pila de 128.
El módulo tiene los siguientes parámetros, entradas y salidas:

```
parameter DATO_WIDTH;
parameter FIFO_LENGTH;
```

```
module (wclk, datin, rclk, rst, dataout, full, empty, dato);

      input wclk;
      input rclk;
      input rst;
      input [DATO_WIDTH-1:0] datin;
      output reg full;
      output reg empty;
      output reg dato;
      output reg [DATO_WIDTH-1:0] datout;
```

Donde:

- `wclk`: Es el reloj de escritura, se activa cuando se desea escribir un dato en la pila de datos.
- `rclk`: Es el reloj de lectura, se activa cuando se desea leer un dato de la pila de datos.
- `rst`: Permite reiniciar la FIFO, todos los registros de salida se vuelven 0 excepto por `empty` que se pone en 1.
- `datin`: El dato de entrada que se escribe en la pila.
- `full`: Su valor es 0, cuando la pila está llena su valor es 1 y no permite escribir más en la pila.
- `empty`: Su valor es 1 cuando la pila no tiene datos, 0 cuando existe al menos un dato en la pila.
- `dato`: Su valor es 1 si existe al menos un valor en la pila, indica si hay datos en la pila.
- `datout`: El dato de salida que se lee de la pila.



Puesto que recibe dos señales de reloj, una para la escritura y otra para la lectura, se puede tanto escribir como leer simultáneamente.

Wishbone:
Para el wishbone del PWM se utilizó la misma lógica que maneja el módulo UART con el que ya cuenta el LM32. Como pwm.v cuenta con lectura y escritura, según el procesador nos pida (wb_rd) o de(wb_wr) información habilitamos estas entradas wr y rd. Para la caracteristica a modificar utilizamos wb_adr_i en adrs. Y por último si es lectura o escritura, damos el valor de dout a wb_dat_o, o a din el valor de wb_dat_i.

Timer
Se usa el módulo timer que viene por defecto en el LM32, el cual se encarga de generar bases de tiempo precisas, del cual se usó la función `msleep(uint32_t msec)`, la cual genera una interrupción en el procesador de `msec` milisegundos.

4. Software

Para la implementación en software del PWM se crearon 3 clases en C++. Estas son: face, que hereda de arm, arm que hereda de pwm y pwm.

Pwm

En esta clase es donde se crean los registros que va a utilizar el wishbone del PWM, como en0, period0, duty0, en1, period1… El mapa de memoria completo se muestra en la siguiente tabla. Donde se tienen todos los registros como solo lectura, ya que no se llegó a implementar por completo la lectura de estos.

READ	WRITE	ADDRESS
	duty7	00 5C
	period7	00 58
	en7	00 54
	duty6	00 50
	period6	00 4C
	en6	00 48
	duty5	00 44
	period5	00 40
	en5	00 3C
	duty4	00 38
	period4	00 34
	en4	00 30
	duty3	00 2C
	period3	00 28
	en3	00 24
	duty2	00 20
	period2	00 1C
	en2	00 18
	duty1	00 14
	period1	00 10
	en1	00 0C
	duty0	00 08
	period0	00 04
	en0	00 00

Esta clase cuenta con cuatro funciones implementadas:

```
pause()
```

```
pwm_en(int sel,uint32_t val)
pwm_period(int sel,uint32_t val)
pwm_duty(int sel,uint32_t val)
```

-`pause` que es un derivado del módulo timer, la cual crea una espera de 1 segundo.
-`pwm_en` que asigna un valor(val) al habilitador de la señal(sel) de PWM.
-`pwm_period` que asigna un valor(val) al periodo de la señal(sel) de PWM.
-`pwm_duty` que asigna un valor(val) al ciclo útil de la señal(sel) de PWM.

arm

Esta clase hereda de la clase pwm. Esta se encarga de generar movimientos completos para los brazos. Como lo puede ser girar la "mano" a la derecha, desplazarla hacia atrás, poner la mano en medio y desplazarla de nuevo hacia adelante, o un simple movimiento de brazo a la derecha. Además de contar con una función para calibrar cada uno de estos movimientos.

- *`init()`:* Lleva los servos de a la posición inicial de agarre del cubo, es decir envía todos los brazos hacia atrás uno por uno, a continuación las manos se ubican en la posición central, y luego los envía hacia adelante uno por uno.
- *`home()`*: Lleva todos los servos hacia atrás uno por uno.
- *`derecha()`:* Rota la mano hacia la derecha lleva el brazo hacia atrás, luego rota la mano a la posición central y termina llevando el brazo hacia adelante.
- *`izquierda()`*: Rota la mano hacia la izquierda lleva el brazo hacia atrás, luego rota la mano a la posición central y termina llevando el brazo hacia adelante.
- *`derecha0()`:* Rota la mano hacia la derecha. Se usa para cambiar el cubo de cara ya que no emplea un tiempo de retardo y se puede utilizar simultáneamente con otra función.
- *`izquierda0()`:* Rota la mano hacia la izquierda. Útil para cambiar la cara del cubo.
- *`medio0()`*: Ubica la mano en la posición central. Útil para cambiar la cara del cubo.
- *`atras0()`:* Lleva el brazo hacia atrás. Útil para cambiar la cara del cubo.
- *`adelante0()`:* Lleva el brazo hacia adelante. Útil para cambiar la cara del cubo.
- *`adelante01()`:* Lleva el brazo hacia adelante. Se usa para variar la fuerza con la que sostiene el cubo cuando lo rota para evitar que este se deslice y se caiga.
- *`set_serv()`:* Se establece el conjunto de servos que componen cada brazo.
- *`calib()`:* Permite cambiar el ciclo útil de la señal de los servos, para calibrar los movimientos del brazo. Se maneja las siguientes direcciones:

der	0x20
iz	0x21
mid	0x22
atr	0x23
ade	0x24
der1	0x25
iz1	0x26
mid1	0x27
atr1	0x28
adel	0x29

Tabla 2. Direcciones para calibrar los brazos.

face

Esta clase que hereda de la clase arm, cumple con la función de generar movimientos útiles con los 4 brazos para el reconocimiento de los colores o la resolución del cubo. Como girarlo con respecto al eje Y, X o Z.

Face1(arm a, arm b, arm c, arm d, uint32_t x)

- *`Face1():`* Este pone el cubo en la posición correcta para tomar las fotos. Va por paso que se ingresan en x, siendo el paso 1 la posición inicial(no hace nada) y el 7 el último paso.

main.c

Es la función principal es ella se establecen cuales servos componen los brazos, y la cual recibe por medio de UART el set de instrucciones y las ejecuta a través de las demás clases y módulos. Tiene la función `instruccion(arm y, char x)`, la cual recibe un brazo y dependiendo del valor de la variable x, `0x10` para ejecutar el proceso de mover la mano hacia la derecha o `0x11` para la izquierda.

Mueve el brazo correspondiente al set de instrucciones que se le envía desde Python, se utilizan las mismas direcciones que en el programa: `Ra, R', La, L', Fa, F', Ba, B', Ua, U', Da, D'.`

Como se comento anteriormente recibe una serie de instrucciones por medio de UART, a continuación se presenta qué acción ejecuta cada comando.

FF F0	Envia los brazos a la posición home.
FE F0	Envia los brazos a la posición init.
FD WX YZ F0	WX = 0x01 mueve el brazo 1 0x02 mueve el brazo 2 0x03 mueve el brazo 3 0x04 mueve el brazo 4 YZ = 0x10 mueve la mano hacia la derecha 0x11 mueve la mano hacia la izquierda
FC WX ST V F0	Permite calibrar el brazo. WX = 0x01 calibra el brazo 1 0x02 calibra el brazo 2 0x03 calibra el brazo 3 0x04 calibra el brazo 4 ST = 0x20 derecha 0x21 izquierda 0x22 medio 0x23 atrás 0x24 adelante 0x25 derecha1 0x26 izquierda1 0x27 medio1 0x28 atrás1 0x29 adelante V = Valor del ciclo útil del PWM.
FB PR F0	Ubica el cubo en la cara PR
FA AD DR F0	Mueve el brazo de acuerdo a la instrucción DR = 0x27 Izquierda 0x32 Derecha AD = 0x52 (R en ascii) 0x42 (B en ascii) 0x4C (L en ascii) 0x46 (F en ascii) 0x55 (U en ascii) 0x44 (D en ascii)
50 F0	Reinicia la cámara.
5A F0	Toma la foto.
5B F0	Obtiene el tamaño de la foto.
5C F0	Envia por UART el archivo JPEG.
5D F0	Deja de tomar fotos.