文章编号:1671-4598(2019)09-0095-05 DOI:10.16526/j.cnki.11—4762/tp.2019.09.021 中图分类号:TP23 文献标识码:A

基于 STM32 的步进电机多轴速度控制方法研究与实现

王昊天[1,2,3]，于乃功[1,2,3]
(1. 北京工业大学 信息学部，北京 100124；2. 计算智能与智能系统北京重点实验室，北京 100124；
3. 数字社区教育部工程研究中心，北京 100124)

摘要：在机器人多轴电机控制过程中，发现带载情况下如果电机起步速度过快会导致电机堵转问题，很需要一种可以实现电机匀加速的精确控制方法；文章借助于 STM32F103，通过其 I/O 口输出矩形波脉冲序列的方式控制步进电机驱动器或伺服驱动器，从而实现对步进电机的位置和速度控制；通过修改定时器值实现梯形加减速轨迹，使步进电机运行具有较好加减速性能；另外，由于 STM32F103 芯片具有高速定时器，可以通过配置定时器输出和插补运算相结合方法，实现对多轴（多个电机）的控制；该方法对于嵌入式步进电机控制器的开发具有很好的参考价值。

关键词：机器人；定时器；多路脉冲输出；梯型加减速算法；步进电机控制器

Research and Implementation of Multi—axis Speed Control Method for Stepping Motor Based on STM32

Wang Haotian[1,2,3], Yu Naigong[1,2,3]
(1. Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China;
2. Beijing Key Laboratory of Computational Intelligence and Intelligent System, Beijing 100124, China;
3. Digital Community Ministry of Education Engineering Research Center, Beijing 100124, China)

Abstract: In the process of motor controlling robot, it is found that, under the load condition, it will cause the motor to stop running provided that the motor starts too fast, based on this, it is necessary to develop a method being able to precise control the uniform acceleration. The paper, with the help of STM32F103, achieved the control against stepper motor driver or servo driver by means of its I/O port outputting rectangular wave pulse sequence, so as to control the position and speed of the stepper motor. By modifying the timer value, the trapezoidal acceleration and deceleration trajectory was realized, in this way, the stepping motor could achieve better acceleration and deceleration performance. In addition, since STM32F103 chip was equipped with multichannel timer, which can be combined by configuring timer output and interpolation operation, so as to achieve the robot control with multi—axis (multiple motors). Such method has a good reference value for the development of embedded stepper motor controller.

Keywords: robot; timer; multi—channel pulse output; trapezoid acceleration and deceleration algorithm; stepper motor controller

0 引言

随着自动化设备和机器人需求的稳步增长，作为它们的关键驱动部件步进电机或伺服电机配套的驱动器及脉冲控制器需求也相应增加，而常用的脉冲控制器一般情况下依靠 PLC 即可实现，但其在机器人控制中无法灵活使用，所以很有必要开发一款基于 STM32 的实时定时脉冲发生器。并且 STM32F103 芯片也有结构简单[1]，成本低廉，占用空间小等诸多优点。

收稿日期:2019-04-08； **修回日期**:2019-06-14。
基金项目:国家自然科学基金项目(61573029)。

作者简介:王昊天(1998-)，男，北京人，主要从事自动化方向的研究。
于乃功(1966-)，男，山东潍坊人，博士研究生导师，教授，主要从事计算智能与智能系统、机器人学与机器人技术、机器视觉领域方向的研究。

电机起步速度过快时会发生堵转，具体原因是因为由静止状态到动态，如果速度过高的话，会引起各轴之间产生冲击，超程，失步等现象[2]，而停止时因为工件在快速运行状态，若突停的话，因机械惯性较大，严重的话会引起机械损伤，或定位不准现象为了使执行机构能平稳定位，就要求电机在开机速度达到给定进给速度的过程中有一个加减速过程，使其能平滑过渡，避免电机速度突变给其带来损伤。

大多数运动控制系统都采用两种加减速控制算法：梯形加减速算法，S 形加减速算法。因梯形加减速方案[2]便于计算，实现方式简单，系统响应快，已能满足一般多轴加减速控制场合应用需求所以在多轴脉冲控制器中获得广泛应用，本文主要采用梯形加减速方法实现。

STM32F103 在发送多路脉冲方面，如果选择用多个定时器来发送，那么在速度非常高的时候就会导致发送的脉

冲不准确。这是因为 STM32F103 芯片是单核单线程的，无法同时处理多个中断，利用多个定时器来发送多路脉冲经常就会发生同一时间触发多个中断的情况。因此本文克服了常规采用多路定时器方案只能实现中低速下（50 kHz 以下）多路脉冲发生方法，采用只使用一个定时器发送主轴脉冲，从轴脉冲通过插补算法跟随主轴的方法实现多路高速脉冲发送。

1 基于脉冲定时器的电机加减速脉冲产生算法

1.1 脉冲定时器工作原理分析

定时器采用向上计数模式，从零开始累加到设定值后溢出，设定值由自己设置并存放在 ARR（自动重装载寄存器）中[3]。当达到设定值时进入中断，每次进入定时中断后，将单片机某一 I/O 口翻转，即可产生脉冲，通过改变定时中断的时间即 ARR 值，就可以控制输出脉冲的频率。为了防止 ARR 寄存器中的值超过 16 位导致溢出，利用 TIMx _ PSC（预分频寄存器），预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。它是基于一个（在 TIMx _ PSC 寄存器中的）16 位寄存器控制的 16 位计数器。这个控制寄存器带有缓冲器，它能够在工作时被改变。新的预分频器参数在下一次更新事件到来时被采用。

当预分频器的参数从 1 变到 2 时，计数器的时序图如图 1 所示。

1.2 插补原理分析

插补法常见的有：逐点比较插补法，比值积分法和数字积分插补法。

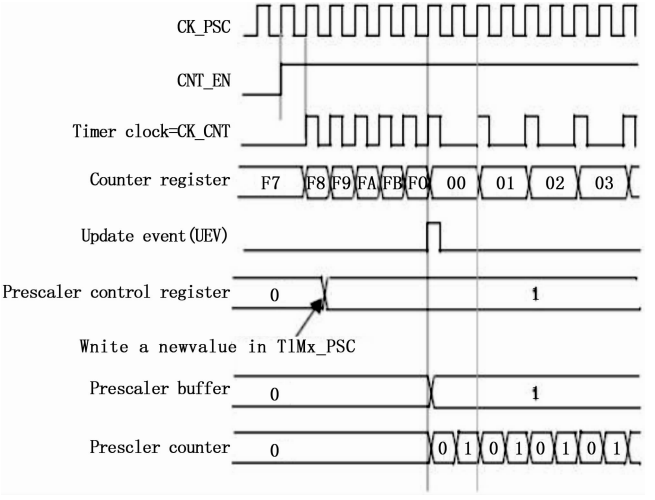

图 1 计数器的时序图

当预分频器的参数从 1 变到 4 时，数器的时序图如图 2 所示。

本文的插补方法主要采用数字积分法，数字积分法就是把给定的形成数据存储到有限长度的寄存器里进行微分累加，通过判断寄存溢出产生脉冲作为进给输出脉冲。数字积分的插补方法（DDA）具有逻辑强的特点，可以实现复杂曲线的插补运算，适用于多轴联动控制；只要输入几

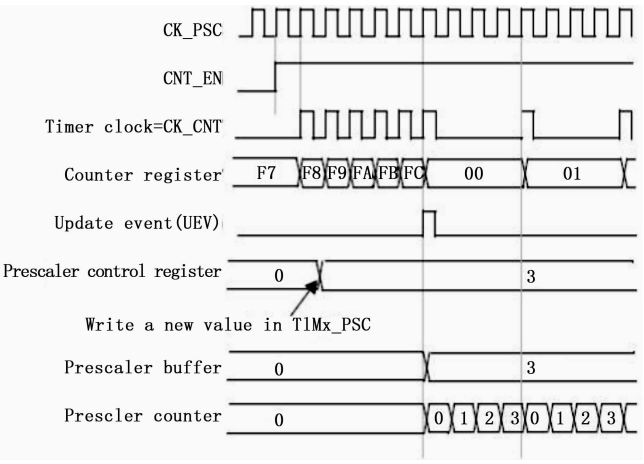

图 2 计数器的时序图

个初始数据，就能计算出执行机构所需要的运行轨迹数据，从而加工出直线、圆弧或由直线和圆弧组成的更复杂的轮廓曲线。在进行插补时选择位置值大的作为长轴，这样长轴就能均匀输出进给脉冲，其余轴就能根据与长轴的关系输出进给脉冲。另外在进行直线插补运算控制时，插补运算的输出脉冲比较均匀。

1.3 电机加减速控制脉冲产生算法

步进电机的转速和步进电机驱动器接收的脉冲频率成正比，如果控制器发出的脉冲频率越高则步进电机的转速越快[4]。利用这个特点通过设定发出脉冲的频率控制步进电机的转速是一种行之有效的方法。通过更改定时器 ARR 的值即可控制输出脉冲的速度[5]。

$$S=\begin{cases} v_0\,t+\dfrac{1}{2}\,at^2 \\ S_1+v_{\max}(t-t_1) \\ S_2+v_{\max}(t-t_2)-\dfrac{1}{2}\,a(t-t_2)^2 \end{cases} \tag{1}$$

脉冲加减速采用常用的梯形加减速方式，参数为初始速度 v_0，最终速度 $v_m$ax，加速度 $acctime$，脉冲总数 L。当这些参数确定后可以确定一个脉冲序列及速度序列（定时器 ARR 值），以 1 ms 为一个单位来改变定时器的装载值，来改变其频率。程序中将每毫秒应输出的脉冲数和该毫秒内定时器的自动装载值都放在查找表中。定时器工作时按照顺序首先按照最初的装载值定时中断，当输出完对应装载值设定的脉冲后，更新装载值。如此复，直到整个加减速过程完成。

计时器初始化参数有如下公式：

$$Tout=\frac{(arr+1)*(psc+1)}{Tclk} \tag{2}$$

那么：

$$\frac{72\ \text{Mhz}}{arr+1}*(psc+1)=1\ \text{ms} \tag{3}$$

定时器装载值：

$$arr=\frac{72000}{psc+1}-1 \tag{4}$$

因为定时器的频率应该是脉冲频率的两倍，所以实际的定时器装载值为：

$$arr = \frac{36000}{psc+1} - 1 \tag{5}$$

1.4　多路脉冲差补算法

确定最高速脉冲为主轴，其他路为从轴[5]。每次中断开始计算其他从轴是否发送脉冲。主轴速度与从轴速度为已知量，主轴速度由 *Vmain* 表示，从轴速度用 *Vsub* 表示。计算方法如下：

$$sy = \frac{Vmain}{2}(\text{向上取整}) \tag{6}$$

$$st = Vmain - Vsub \tag{7}$$

如果 $sy \geqslant st$ 那么发送脉冲，并且：

$$sy = sy - st \tag{8}$$

如果 $sy < st$ 那么不发送脉冲，并且：

$$sy = sy + Vsub \tag{9}$$

如此往复循环就可以得到从轴脉冲。

2　步进电机速度与位置控制的实现

2.1　梯形加减速情况

步进电机单轴速度与位置控制程序流程图如图 3 所示。首先，设置步进电机的初始转速 v_0，最高转速 v_{max}，加速时间 *acctime*，总脉冲数 *L*。然后，计算出加速段截止脉冲数 *S*1，匀速段截止脉冲数 *S*2。根据不同区间的位置 *S* 使用不同的赋值方式[6]。接着利用上述给定条件，计算出每毫秒发送的脉冲数与截止至该毫秒的总脉冲数（每毫秒发送的脉冲数实际上是只有加速阶段的速度值并且是经过 1.3 节算法计算出的定时器装载值），并设置两个数组分别装载所有计算结果。接着每毫秒都先进行判断，首先判断总脉冲值是否小于 *S*1，如果小于则说明目前为加速阶段，将数组中的每毫秒发送脉冲数按正序赋值给定时器，如果不再满足小于 *S*1 的条件，则开始循环判断是否小于 *S*2，如过小于则说明目前在匀速段，将数组中每毫秒发送的脉冲数中最大值赋给定时器，如果不再满足小于 *S*2 的条件，则开始循环判断是否小于 *L*，如果小于说明目前为减速阶段，将数组中的每毫秒发送脉冲数按倒序赋值给定时器，如果不再满足小于 *L* 的条件，说明脉冲已全部发送完毕。采用先计算再用数组存储的方法是为了使输出脉冲具有连续性与准确性，并能够减少 CPU 占用。

2.2　三角形加减速情况

在输入完设定值后，如果无法在总脉冲为 L 时完成一次完整的从 v_0 加速到 v_{max} 再减速回 v_0 时，说明 $v_m$ax 设置值过大。而实际的发送情况是加速到发送脉冲总数为 1/2*L* 就立刻开始减速，实际最大速度应为 v_1，所得速度函数应为三角形而非梯形。考虑到此情况，当按梯形加减速求出的 $S1 > 1/2L$ 时，则不使用设置的 $v_m$ax 为最大速度而利用 1/2*L*、v_0、*acctime* 求出实际最大速度 v_1 替换掉梯形加减速情况的 $v_m$ax，1/2*L* 替换掉 *S*1，取消掉匀速段部分 *S*2 的判断与赋值，即可实现准确的三角形加减速。

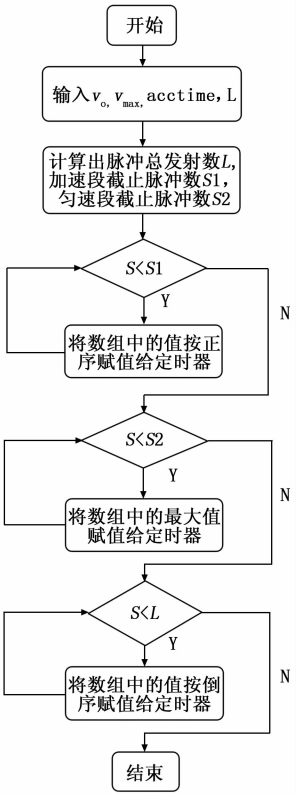

图 3　单轴脉冲加减速流程图

实际代码：

（1）主程序配置代码如下：

```
ifndef _DRIVE_H_
define _DRIVE_H_
include "sys.h"

define PULSEARRAY_SIZE 500
extern u16 psc;
struct PULSE
{
    u16 pulsearray[PULSEARRAY_SIZE];  //存放每毫秒要发送的脉冲数,脉冲频率
    u32 pulse_num;  //定时器中对发送脉冲计数
    int pulsearray_index;  //中间变量,用于定时器值累加
    u16 state_status;  //运行状态标志
    u16 state1;  //S1 分割点
    u16 state2;  //S2 分割点
    u16 state3;  //L 结束
    u16 time_up;  //加速时间 acctime
    u16 time_stable;  //平稳运行时间
    u32 pulse_num_stable;  //平稳运行时的脉冲总数
    u16 v0;  //初始速度
    u16 vmax;  //最终速度
    float acc;  //加速度
    u32 pulse_totalnum;  //脉冲总数
};
```

```
extern struct PULSE pulse_x;
extern struct PULSE pulse_y;
extern struct PULSE pulse_z;
extern struct PULSE pulse_a;
extern struct PULSE pulse_b;
extern struct PULSE pulse_c;
u8 Init_Pulsetable(u16 v0,u16 vmax,u16 acc,u32 pulse_totalnum,struct PULSE * pulse);//声明使用的速度计算函数
  endif / * __DRIVE_H__ * /
```

(2) 主程序代码如下：

```
u8 Init_Pulsetable(u16 v0,u16 vmax,u16 acc_time,u32 pulse_totalnum,struct PULSE * pulse)
{
u16 i = 0;
u32 pulse_num_temp = 0;
//先判断速度是否满足要求
if(v0 < 1)v0 = 1;
if(vmax > 100)
vmax = 100;
pulse->v0 = v0;
pulse->vmax = vmax;
pulse->time_up = acc_time;
pulse->pulse_totalnum = pulse_totalnum;
pulse->state_status = 1;
pulse->acc = (float)(vmax-v0) / acc_time * 1.0;
if(pulse->time_up * 2 + 2 > PULSEARRAY_SIZE) return 0;
pulse_num_temp = (v0 + v0 + pulse->acc * (pulse->time_up-1)) * pulse->time_up;
if(pulse_num_temp < pulse->pulse_totalnum)
{
//梯形加减速
for(i = 0;i < pulse->time_up-1;i++)
{
pulse->pulsearray[2 * i] = v0 + pulse->acc * i;
pulse->pulsearray[2 * i+1] = 36000 / pulse->pulsearray[2 * i] / (psc+1)-1;
}//将每毫秒计算出的自动重装载值存入数组
pulse->pulsearray[2 * pulse->time_up-2] = pulse->vmax;
pulse->pulsearray[2 * pulse->time_up-1] = 36000 / pulse->pulsearray[2 * pulse->time_up-2] / psc / 2;//将实际算出的脉冲总数存入数组
pulse_num_temp = (v0 + v0 + pulse->acc * (pulse->time_up-2)) * (pulse->time_up-1);
pulse_num_temp += pulse->vmax * 2;
pulse->pulse_num_stable = pulse->pulse_totalnum-pulse_num_temp;
pulse->time_stable = pulse->pulse_num_stable / vmax;
pulse->state1 = pulse->time_up;
}
else
{
//三角形加减速
pulse->time_up = (sqrt(4 * v0 * v0+4 * pulse->acc * pulse_totalnum)-2 * v0)/(2 * pulse->acc);
pulse_num_temp = (v0 + v0 + pulse->acc * pulse->time_up) * (pulse->time_up + 1);
pulse->pulse_num_stable = pulse->pulse_totalnum-pulse_num_temp;
for(i = 0;i <= pulse->time_up;i++)
{
pulse->pulsearray[2 * i] = v0 + pulse->acc * i; //将实际算出的脉冲总数存入数组
pulse->pulsearray[2 * i+1] = 36000 / pulse->pulsearray[2 * i] / (psc+1)-1;
//将每毫秒计算出的自动重装载值存入数组
}
pulse->state1 = pulse->time_up+1;
}
return 1;
}
```

3 多路脉冲发送实现

多路脉冲发送流程图如图 4 所示。由于 STM32F103 性能不够强大，不能够同时使用多个定时器同时发送多路高速脉冲，所以选择最高速脉冲为主轴[9]，用定时器来控制，其余各路作为从轴用插补算法来控制发送速度，从而避免使用多个定时器。流程是先判断是否发送，然后进入中断，通过比较找出最高速的一路脉冲作为主轴，其余路脉冲为从轴，然后通过差补算法算出从轴在此次中断时是否发送脉冲。如此不断重复即可实现单个定时器实现多路不同速度脉冲的发送。实际上，在主函数中也需要先判断一次最高速脉冲，然后才能确定主轴与中断触发条件。实现从轴发送与不发送脉冲的方法是不断改变一个设定好的全局变量标志位，目标从轴的标志位为 1 时从轴发送，为 0 时不进行发送，在主函数里则通过判断这些标志位来判断是否发送，这样是因为中断函数里不能出现循环语句导致的。

4 测试与实验结果分析

4.1 单路加减速脉冲实现

将按照要求实现的程序烧录进 STM32F103 开发板中，输入初始速度，最大速度，加速时间 acctime，总步长 L。然后开始发送脉冲并用示波器测量波形，得到 3 个阶段的波形如图 5、图 6 和图 7 所示。可以发现由控制 I/O 口发送脉冲的方法更加精准且更易于计数。在实际自动化生产与机器人运用中，精确的速度控制往往能够成为是否可应用化的关键标准，所以开发以一款基于 STM32F103 由控制 I/O 口发送脉冲的方法是十分必要的。

4.2 多路脉冲发送实现

在原有程序基础上加入多路脉冲插补算法重新烧录进 STM32F103 开发板中，设定主轴脉冲最高速度为 50 kHz，

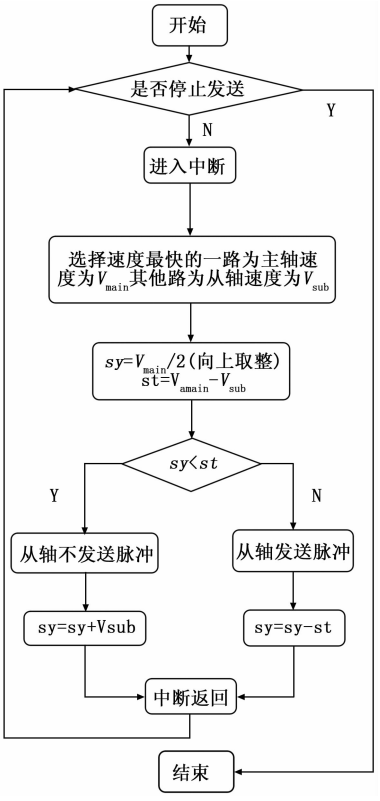

图 4　多路脉冲发送流程图

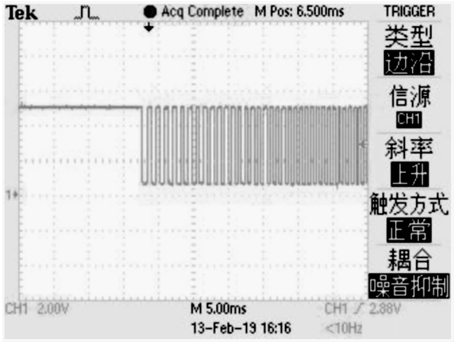

图 5　脉冲发送加速段

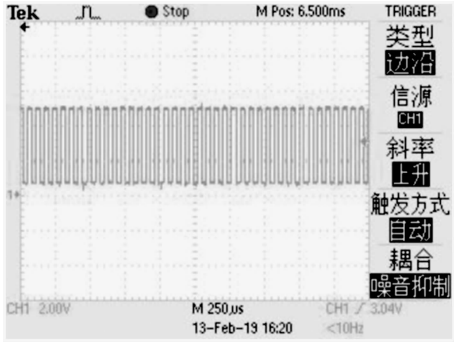

图 6　脉冲发送匀速段

其余两路分别为 30 kHz 与 20 kHz。得到的三路脉冲波形图如图 8 所示，图 9 为由 3 个定时器发送三路脉冲的方案，通过对比可以看出由差补算法得到的速度比为精确的 5：3：2，而利用 3 个定时器发送的方式主轴实际速度为 30 kHz，无法达到 50 kHz 且速度不稳定，这是由于 STM32F103 本身性能不够的原因导致的，所以使用插补算法发送多路脉冲是十分有必要的。

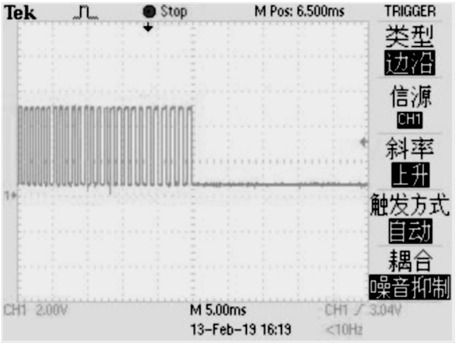

图 7　脉冲发送减速段

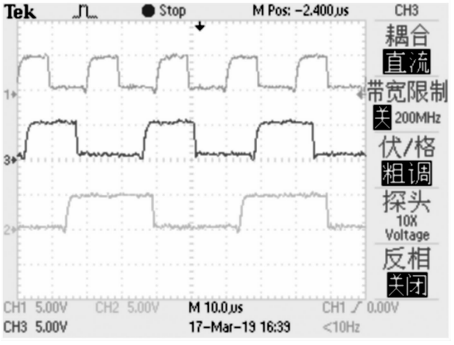

图 8　利用差补算法方案的多路脉冲

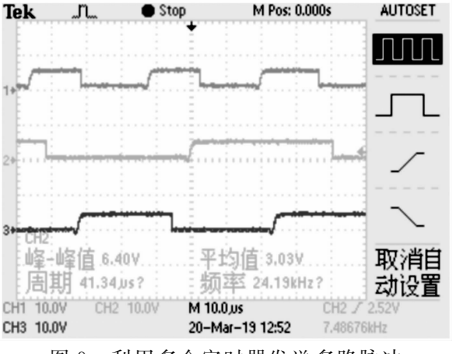

图 9　利用多个定时器发送多路脉冲

5　结束语

本文中设计的基于 STM32F103 的控制多轴梯形加减速脉冲发送方法具有精度高，硬件结构简单，适应性强等特点[10]，可降低运动控制器的成本。该方法已经应用于工业自动化控制领域，证明了该方法的可行性与实用性。

参考文献：

[1] 赵成龙，张春雷，陈　龙. 基于定时器的步进电机控制程序设计 [J]. 精密制造与自动化，2018 (4)：30－32.

[2] 杜文广. 基于 FPGA 多轴运动控制专用芯片的研究 [D]. 西安：西安工程大学，2012.

（下转第 108 页）